


高等学校计算机应用型本科规划教材精选



Android 移动应用程序开发教程



朱凤山 编著
王慧芳 主审

清华大学出版社

21世纪高等学校计算机应用型本科规划教材精选

Android移动应用程序开发教程

朱凤山　编著
王慧芳　主审

清华大学出版社
北京

内 容 简 介

本书主要介绍 Android 平台移动应用程序开发的知识，从基础知识开始讲解，由易入难，循序渐进，系统地介绍了 Android 应用程序开发中所用到的知识。注重引导学生掌握开发技巧，理解处理问题的思路，培养学生分析问题、解决问题的能力。本书可作为高等院校计算机、软件工程及相关专业的本、专科学生学习 Android 移动平台应用程序开发的教材，也可供该领域的教师、开发人员学习研究 Android 移动平台应用程序开发时参考。

图书在版编目（CIP）数据

Android 移动应用程序开发教程/朱凤山编著. —北京：清华大学出版社，2014
21 世纪高等学校计算机应用型本科规划教材精选
ISBN 978-7-302-35978-4

Ⅰ. ①A…　Ⅱ. ①朱…　Ⅲ. ①移动终端—应用程序—程序设计—高等学校—教材　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2014)第 066053 号

责任编辑：索　梅　薛　阳
封面设计：杨　兮
责任校对：梁　毅
责任印制：

出版发行：清华大学出版社
　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
　社 总 机：010-62770175　　**邮　　购**：010-62786544
　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
　课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　张**：18.75　**字　　数**：468 千字
版　　次：2014 年 5 月第 1 版　**印　　次**：2014 年 5 月第 1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：051835-01

前 言

FOREWORD

Android是一种基于 Linux 内核、开放源代码的操作系统，主要使用于移动设备，如智能手机、平板电脑、电视等。来自互联网的统计数据显示，Android 已经成为目前使用最为广泛的移动操作系统，远超 Apple 公司的 iOS 和 Microsoft 公司的 Windows Phone。根据 Gartner 对智能手机操作系统占有市场份额的预期，到 2015 年 Android 操作系统的占有份额将达到 50%左右，远高于其他操作系统。

对于学习 Java 编程语言的读者，Android 操作系统的出现，提供了新的学习方向。巨大的市场需求，提供了更多的机会，也急需更多的开发者提供更加丰富的应用。本书主要针对学习过 Java 编程语言，具备一定的编程基础，有意愿学习 Android 平台应用程序开发的读者人群。如果你目前正处于这种状态下，本书比较适合你的选择。

本书在编写过程中，按照知识的逻辑关系分章，循序渐进、重点突出，对知识点的讲解与介绍做到尽量全面，并给出可以应用于何种场合的建议。对于重、难点知识，给出专门的演示项目，按步骤讲解实现方式。多数学习开发的读者在熟悉了语法知识之后，都想迫不及待地一展身手，编写一款自己的软件，虽然这是良好的学习习惯，也是值得肯定的学习编程的积极态度。但是，如果所选择的项目过大、过于复杂，往往很难将功能实现，即使有参考代码和帮助文档，也会陷入代码海洋或文档风暴中，这样只会收到事倍功半的效果，而且，学习的积极性也会受到很大的打击。所以，对于初学者，建议选择功能单一、结构简单的项目。

全书所有章节讲解知识的方式统一，章节结构清晰，方便读者快速查询相关问题。每个章节开始都给出了该章的主要内容，列举出该章主要介绍的知识点。在介绍内容时，根据不同知识点的具体情况，介绍知识点的分类、周边信息并总结功能实现的步骤。书中经常涉及以下符号，读者应了解其含义：

- 表明知识点涉及内容的划分或实现步骤的说明。如对某个类中常用方法的说明，实现某个功能可以采用的方式有哪些。该符号所对应的知识点都需要掌握、熟记，能够达到应用的水平。
- 关键功能代码。该符号对应一段能够实现某个功能的关键代码，考虑到篇幅问题，多数情况下，它们并不是一段完整的代码，如果需要阅读项目的所有源代码，可查阅本书配套电子资料。
- 对某些知识在运用到项目中的建议，或者在使用某些知识时应该注意的事项。

本书配套电子资料中含有本书中所列全部项目（WorkSpace 是 BookDemo），读者可以将整个工作空间都引入 Eclipse 中。不同章节项目的命名以 part××区分，同一个章节中不同项目的命名以 part××_×区分。读者应特别留意命名为 part××_×_×的项目，这些项目是某些 Android 项目所依赖的 Web 端程序，主要是第 10 章的项目。配套资料中的

Android 项目在开发时采用 Eclipse ADT4.2、SDK4.2,运行的目的平台最低是 Android 2.3.3。配套资料中的 Web 项目在开发时采用 MyEclipse 9.1,这些项目需要部署在服务器中(如 Tomcat 6.0)才能运行。

全书共分为 10 章。第 1 章介绍 Java 语言的基础知识;第 2 章介绍 Android 开发环境,Android 项目的结构,Android 项目的调试与发布,并详细介绍了 Activity 的使用;第 3 章介绍 Android 平台 android.wigdet 包中基本控件的使用,包括 Form Widgets 控件、TextFields 控件、布局管理器、Image 和 Media 控件、Time 和 Date 控件等;第 4 章介绍 Android 平台高级控件的使用,包括 ListView 与适配器的使用、ExpandableListView、GridView、ScrollView 和 HorizontalScrollView、SlidingDrawer、TabHost 和 TabSpec、Gallery 和 ImageSwitcher;第 5 章介绍 Android 平台系统组件的使用,包括 Menu、自定义 Dialog、Notification、ActionBar 的功能解析和布局新方式 Fragment;第 6 章介绍四大组件 Activity、Service、BroadcastReceiver、ContentProvider,以及 Intent 与 IntentFilter;第 7 章介绍 2D 游戏开发的基础知识,包括 View 与 SurfaceView 的使用、Canvas 和 Paint 介绍、如何绘制游戏元素、屏幕坐标与屏幕事件等;第 8 章介绍多媒体操作的内容,包括 MediaPlayer、SoundPool、VideoView 的使用;第 9 章介绍 Android 平台数据存储的三种方式,包括 SharedPreferences 与 Editor 的使用,I/O 流与读写 SD 卡,嵌入式数据库 SQLite 的使用;第 10 章是 Android 平台网络编程的内容,包括 TCP 通信与 Socket 应用、URL 访问网络资源、HttpClient 的应用与 WebService 的应用、XML 解析与 JSON 解析。

作为 developer.android、CSDN、51CTO、eoeandroid、机锋开发者等技术论坛和社区的忠实用户和学习者,在本书的编写过程中,作者从中受益匪浅,也建议读者在遇到学习问题时,可以向专业技术论坛或社区求助。在本书完成之际,特别要感谢王慧芳教授、王志军教授给予的指导和建议,感谢桑婧、王慧、徐峰、王新峰和新锐 IT 工作室的成员给予的启发和帮助,也要感谢张新芳、朱思齐的大力支持。

由于作者学术与经验的欠缺,在本书的结构、知识点与难点的选择和解析过程中,存在一定的问题与不足,希望广大读者不吝赐教。相关技术问题可以发送邮件到 tj_zhufengshan@163.com,只要有时间,作者当尽量给每个人回信。

朱凤山
2014 年 1 月

目 录

CONTENTS



CHAPTER 1

第1章

开发语言与开发环境

本章主要内容：

Java 背景知识；
开发环境配置；
Java 基本数据类型；
Java 基本控制语句；
Java 语言的特点。

1.1 Java 背景知识

Java 起源于 20 世纪 90 年代初 Sun 公司(于 2009 年被 Oracle 公司收购)的一个叫 Green 的项目，该项目的目标是开发嵌入家用电器的分布式软件系统，使电器更加智能和通用(避免针对不同芯片重复编码，即让代码在不同芯片上都可以正常运行)。

1998 年 12 月 4 日，JDK 1.2 隆重发布，标志着 Java 2 平台的诞生。Sun 公司在 Java 1.2 版以后将 JDK 1.2 改名为 J2SDK，将 Java 改名为 Java 2。在 1999 年 Sun 公司还将 Java 2 平台分为三大块：J2EE、J2SE、J2ME，具体说明见表 1.1。

表 1.1　Java 2 的三大平台

名称	说　明
J2EE	Java 2 Enterprise Edition(企业版)，适用于服务器，目前已成为企业运算、电子商务等领域的热门技术
J2SE	Java 2 Standard Edition(标准版)，适用于一般的计算机，开发 PC 上的应用软件
J2ME	Java 2 Micro Edition(微型版)，适用于手持设备，进行应用开发，如手机游戏、名片管理等

在今后的讲解和介绍中，将以 J2SE 6.0 为平台，讨论 Java 语言程序基础知识。J2SDK 1.6 的标准版又名 Java SE 6。读者无须再纠结于 Java 的哪个开发版本，尤其是刚刚接触的读者，无须关注各版本间的差别，建议统一选择 J2SE 6.0 作为自己的学习和开发平台，对于目前最新的 J2SE 7.0 感兴趣的读者可以自己尝试。

不仅平台发生变化，随着供应商的不同，Java 的 API（Application Programming Interface，应用程序编程接口，供程序员调用）被分为三大类，Google 公司对 Android 平台提供了第 4 类 API，这些 API 在后续的学习中都会有所涉及。

➢ Java Core API：由 Sun 公司制定的基本的 API，包名是 Java. ，所有的 Java 平台都应该提供。

➢ Java Optional API：由 Sun 公司制定的扩充 API，包名是 Javax. ，Java 平台可以有选择地提供。

➢ 特殊 API：由特殊厂商或者组织提供的 API，包名是 org. 。

➢ Android 平台所采用的由 Google 公司提供的 API，包名是 android. 。

时至今日，J2SE 已经发展为一个覆盖面广、效率高、易用性强的技术平台，其体系结构如图 1.1 所示。截至目前，读者不要企图搞明白图中的所有技术，在以后的讲解中，也不可能介绍全部的知识，读者要明白一点，J2SE 6.0 已经非常成熟和完善了，提供了强大而丰富的 API，它足以让我们很轻松地完成基于该平台上的所有软件开发工作。

Java Language	Java Language
Tools & Tool APIs	java, javac, javadoc, apt, jar, javap, JPDA, jconsole
	Security, Int'l, RMI, IDL, Deploy, Monitoring, Troubleshoot, Scripting, JVM TI
Deployment Technologies	Deployment, Java Web Start, Java Plug-in
User Interface Toolkits	AWT, Swing, Java 2D
	Accessibility, Drag n Drop, Input Methods, Image I/O, Print Service, Sound
Integration Libraries	IDL, JDBC™, JNDI™, RMI, RMI-IIOP, Scripting
Other Base Libraries	Beans, Intl Support, I/O, JMX, JNI, Math
	Networking, Override Mechanism, Security, Serialization, Extension Mechanism, XML JAXP
lang and util Base Libraries	lang and util, Collections, Concurrency Utilities, JAR, Logging, Management
	Preferences API, Ref Objects, Reflection, Regular Expressions, Versioning, Zip, Instrument
Java Virtual Machine	Java Hotspot™ Client VM, Java Hotspot™ Server VM
Platforms	Solaris™, Linux, Windows, Other

图 1.1 Java 6.0 API 结构

通过上面的介绍读者应该清楚一点，Java 语言比较适合开发互联网应用程序、移动应用程序，如果目标是这个方向的，那么选择 Java 语言是比较合适的。下面给出选择编程语言的几条建议。

- 编程语言目前或将来是否占有主导地位，这样可以保证所开发的产品具有广泛的市场。
- 编程语言是否支持重用，以便于提高软件的开发效率。
- 类库和开发环境是否比较成熟。

1.2 开发环境配置

确定要学习 Java 语言，首先要准备开发环境，就好比练习刀枪需要先准备场地一样。下面分别介绍两种开发环境，一种是 Java SDK 自带的，效率比较低，另一种是集成化的开发环境（Integrated Development Environment，IDE），可以很快地组织、编写和测试代码。

1.2.1　Java SDK的安装、配置与测试

目前使用较多的 SDK 版本是 6.0,在后续编程过程中,建议读者使用 SDK 6.0,便于知识点同步。下面分步介绍如何安装和配置该环境。

(1) 获取安装包。SDK 安装包可以去 Oracle 的官方网站下载,也可以去百度或 Google 网站搜索。需要注意的是,获取的安装文件是否是 Windows 平台兼容的。比如 jdk-6u1-windows-i586-p.exe,就是可以安装在 Windows 平台上的。

(2) 安装包里除了包含 Java 语言编译器、调试器以及演示程序以外,一般还会包含 Java 程序的运行环境(Java Runtime Environment,JRE)。JRE 是某一平台运行 Java 程序的软件环境,包括虚拟机 JVM(Java 程序可以跨平台运行的基础)和核心类库等。安装过程中,会询问是否安装 JRE,选择安装就可以了,建议默认安装到 C 盘(文件大约 70MB)。

(3) 配置 JDK。安装完成后,需要对其进行配置。右击"我的电脑",选择"属性"命令,在"系统属性"窗口中打开"高级"选项卡,单击最下面的"环境变量"按钮,弹出"环境变量"编辑窗口。在"系统变量"面板中单击"新建"按钮,创建如表 1.2 所示的三个环境变量(JDK 默认安装情况下)。

表 1.2　JDK 环境变量

名称	值	说　明
JAVA_HOME	C:\Program Files\Java\jdk1.6.0_13	JDK 根目录
Classpath	.;%JAVA_HOME%\lib;	lib 类库路径,注意有个".",多个值用";"隔开
Path	%JAVA_HOME%\bin;path	bin 目录路径

(4) 测试安装。配置好环境变量后,写一个简单的程序以作测试。建议在 D 盘或 E 盘中建立一个文件夹,命名为 BookDemo,再建立 test01 子文件夹。在路径 E:\BookDemo\Part01 中新建一个文本文件,并复制以下代码。

代码 01-1

```
public class FirstDemo{
    public static void main(String [ ]args){
        System.out.println("Hehe");
    }
}
```

将该文本文件改名为 FirstDemo.Java(如没有后缀名,需要在文件夹选项中开启显示常见文件后缀名),与代码中 class 后面的类名一致。

在"运行"界面中输入"cmd",进入命令行界面,输入"E:",即可进入 E 盘,输入"cd BookDemo\Part01"进入代码所在文件夹。输入命令"javac FirstDemo.java",进行编译,如无错误,该文件夹下会生成一个名为 FirstDemo.class 的文件,如图 1.2 所示。其中.java 文件是源代码文件,刚刚生成的.class 文件是编译后的文件,可以直接运行。



图 1.2 编译后的文件

在命令行界面输入命令“java FirstDemo”，显示“Hehe”，表明代码运行正常，环境变量配置正确，全部过程如图 1.3 所示。

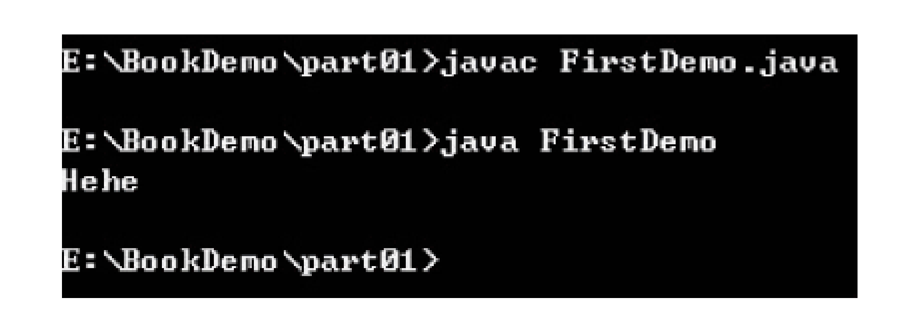


图 1.3 代码 01-1 的编译和运行

如果在测试过程中，没有得到上面的结果，请按以下几条原因排查。

- 代码并非复制，而是自己动手输入的(如果真是这样做的，恭喜你已获取学习程序语言的秘籍了)，请对照代码 01-1，检查是否有拼写错误，Java 语言对大小写是敏感的，务必注意这一点。
- 环境变量配置有误，请对照前面的介绍，仔细检查环境变量的配置。
- 文件命名是否正确，要求文件名称与代码中 class 后面的类名一致，且后缀为.java。

1.2.2 Eclipse 的安装与测试

借助 JDK 自带的 javac 和 java 命令，虽然可以编写程序，但是效率比较低。下面介绍如何借助集成化的开发工具编写代码。支持 Java 开发的集成开发工具有很多，如 JBuilder、Netbeans、WebLogic、Eclipse。在以后的开发过程中，主要使用 Eclipse 开发环境，这也是目前比较流行且成熟的开发工具。

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。最初是 IBM 公司的产品，后又贡献给开源组织 Eclipse.org。2003 年 3 月，Eclipse 2.1 一经发布，以其友好的界面和强大的功能，吸引了众多大公司加入到 Eclipse 平台的发展上来，目前许多实际开发中应用的开源框架如 Spring、Struts、Hibernate 等都会附带提供 Eclipse 的插件支持。

安装、使用 Eclipse 比较简单，可参看以下步骤进行。

(1) 确保 JRE 已正确安装。

Eclipse 需要 Java 运行环境的支持，必须先安装 JRE，如果已经完全安装过 JDK，此步骤可以省略。

(2) 获取 Eclipse 安装包。

安装包可以在 http://www.eclipse.org/downloads/网站下载，由于版本众多，读者要仔细阅读说明，而且要注意 32 位操作系统和 64 位操作系统是不同的。后面讲述使用的是 Eclipse Java EE IDE for Web Developers，版本为 Helios Release。

下载与操作系统匹配的安装包后，直接解压到相应目录即可，Eclipse 是完全绿色的，无须安装。进入解压后的路径，将 eclipse.exe 发送到桌面快捷方式，方便以后打开。

（3）测试 Eclipse。

第一次打开该软件会选择工作文件夹（便于所有创建的项目和代码文件的存放），直接选择前面创建的 workspace 文件夹即可，Eclipse 打开后会显示欢迎界面，直接关闭也可，现在不对该软件一一介绍，随着学习的深入，会慢慢了解到这款软件的强大之处。软件的布局和常用的办公软件差不多，上面是菜单栏和快捷工具栏，中间区域是主要部分，左边是 Package Explorer，以后新建的项目都会罗列在此，中间是主要编辑区域，用于完成代码的编辑，右边是 Outline，显示所编辑区域的类信息。下面还有一些其他视图，这些窗口视图可以在菜单中选择 Window→Show View 命令。目前会用到的是 Console（控制台）视图，用以显示输出信息和输入数据。

选择 File→New→Java Project 命令，输入项目名称“BookDemo”，单击 Finish 按钮，完成创建新项目。展开此项目会有两部分，上面是 Src 文件夹，下面是 JRE。程序代码当然要写在 Src 文件夹中，但是，一般不会直接在 Src 下面建立代码文件，而是先创建一个 package（包），然后将代码文件创建在包中，当然，包下面可以继续创建包，这样的好处是用包来管理代码文件，比较方便，功能类似的代码放在同一个包中，也便于查找。

选择 Src 文件夹，右击，选择 New→Package 命令，新建一个包，输入包名（建议小写），再选择此包，选择 New→class 命令，新建一个类，输入类名。全部完成后，Package Explorer 视图中应该如图 1.4 所示。

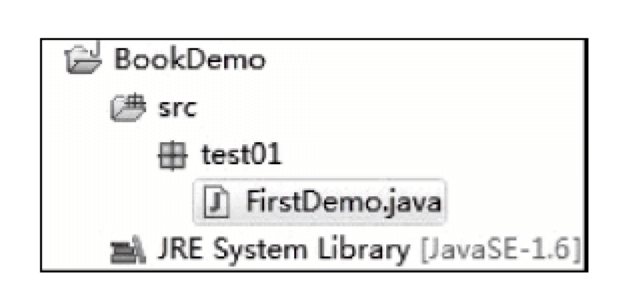


图 1.4　新建 FirstDemo 视图

代码 01-2

```
package test01;                        //包名
public class FirstDemo {               //自动生成的类,class 关键字用于定义类,class 前是修饰词
    //main 方法是代码的主方法,如果代码要运行,需要 main 方法
    public static void main(String args[]){
        System.out.println("Hehe");    //实现向 Console 输出字符串 Hehe
    }
}
```

输入代码 01-2，基本和代码 01-1 一致，只是增加了包名的声明（Eclipse 自动添加），前面已经介绍了代码是用包的方式来进行管理的，希望读者在编写代码时都放在不同的包中。代码编写完成后，在代码编写区域单击右键，选择 Run as→Java Application 命令，如在 Console 视图输出字符串“Hehe”，表明 Eclipse 可以正常工作。关于 Eclipse 的使用和 Java 代码的编写，应遵循以下几条建议。

- 项目的命名、包的命名、类的命名尽量使用字母和数字的组合，且包名一般都小写，类名每个单词的首字母大写（驼峰表示）。
- 书写代码时注意缩进，错落有致，请读者品味以上两段代码的书写风格。
- 代码应该有注释语句，Java 中常用的注释方式有三种，分别是 //、/* */、/**

*/。后两种可以用于多行注释，后面的代码中会用到。

- 在编写代码的同时，Eclipse 已经编译好代码了(编译好的代码放在 bin 目录中，bin 目录是与 Src 对应的一个目录)，所以可以直接运行，不需要向前面介绍的那样先 javac，然后 java，直接运行即可。
- 借助 Eclipse 编写代码，需要先建立一个 class，增加一个 main 方法，然后将要运行的代码写在 main 方法中(main 方法由 JVM 调用)。

1.2.3 搭建 Android 开发环境

到此为止，开发 Java 应用程序的环境已经搭建完毕，在此基础上下面介绍如何搭建开发 Android 应用程序的环境。整个过程稍显烦琐，需要先下载 Android 的 SDK(Software Development Kit，软件开发工具包)，配置环境变量，然后给 Eclipse 安装(“升级”更准确)插件 ADT(Android Development Tools)，并指定 SDK 的位置，最后创建 Android 虚拟设备(Android Virtual Device，AVD)，即虚拟 Android 系统手机。下面给出详细的安装与配置步骤。

1. 下载 SDK

SDK 的版本更新得比较快，获取的方式也在改变，以下描述的获取 SDK 的过程是当时最新的，但无须按部就班地进行，归根结底只要得到 SDK 就可以。Android 开发社区，下载 SDK 的网址是 http://developer.android.com/sdk/index.html，页面如图 1.5 所示，选择合适的开发平台，如果是 Windows 平台就可以选择 installer_r12-windows.exe，这也是 Android 推荐项。具体安装事宜可以阅读该网站的 Installing the SDK，比如系统的需求、软件的需求等。如果采用本书的开发环境，则无须阅读，可以直接向下进行。

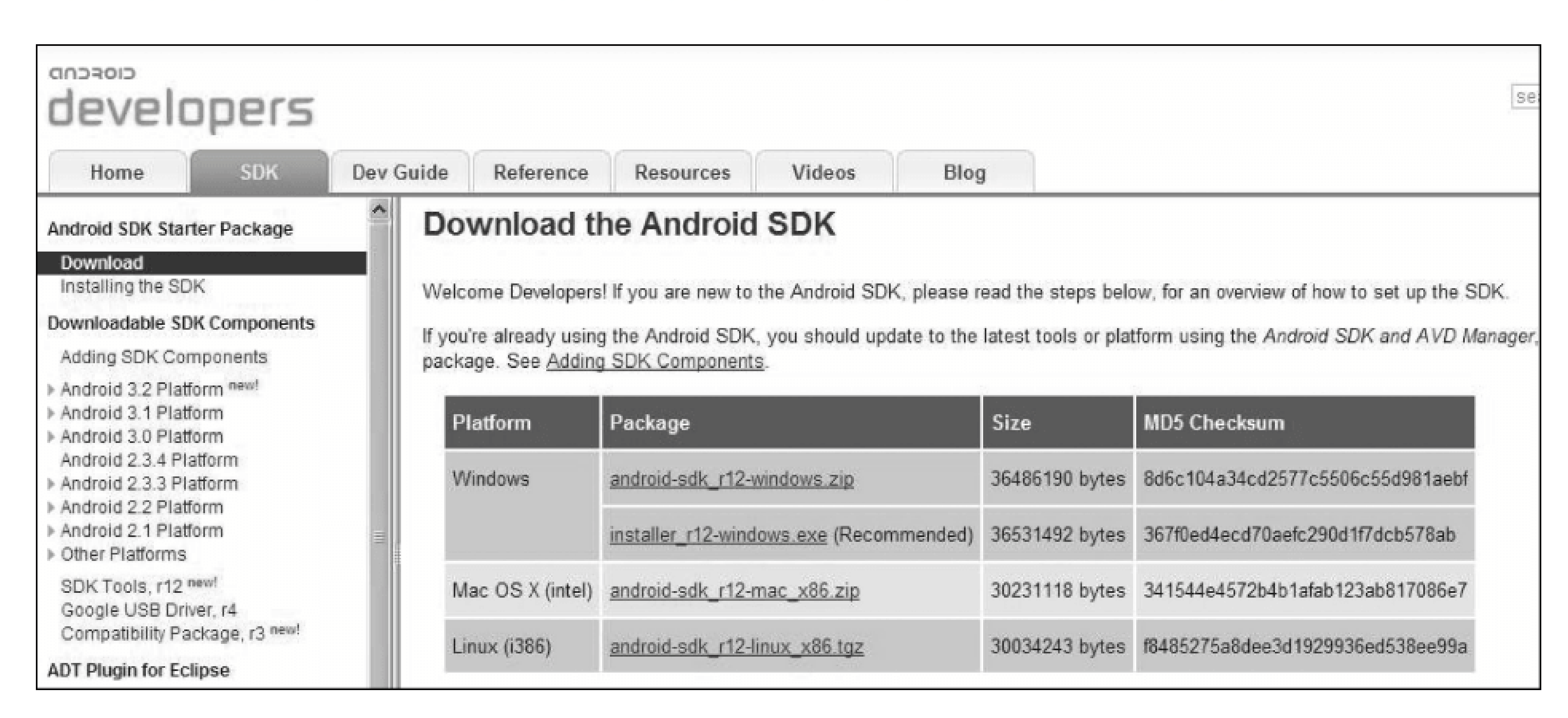


图 1.5 SDK 下载页面

安装结束后，会打开 Android SDK and AVD Manager 窗口，开始更新，选择全部更新即可，更新有时会比较慢，取决于网速，更新界面如图 1.6 所示。更新完成后，将安装路径中的 tools 文件夹增加到环境变量 path 中，在本书中是 E:\Android\android-sdk\tools。检验 SDK 是否安装成功可以在“运行”界面中输入“cmd”，打开命令行窗口，输入“android -h”，如

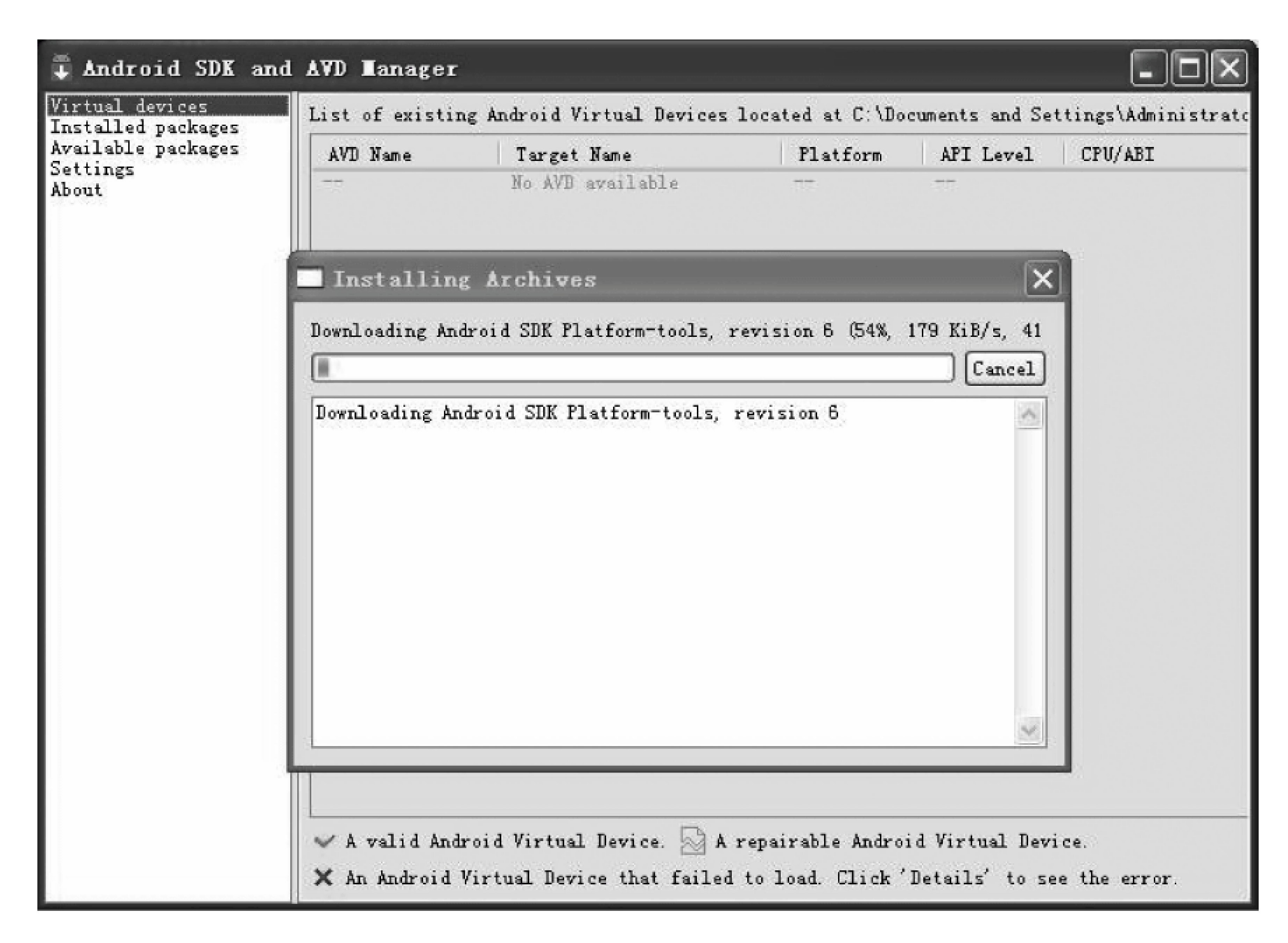


图 1.6 Android SDK and AVD Manager 窗口

果识别,表示安装成功。

2. 安装 ADT

ADT 是借助 Eclipse 开发 Android 应用程序的插件,Eclipse 默认没有该插件,需要手动安装。但是并不是所有 Eclipse 版本都可以安装该插件,需要 Eclipse 3.5 或更高版本。如果采用的 Eclipse 满足该要求,可以继续进行,否则需要下载新版 Eclipse。

打开 Eclipse,在 Help 菜单中选择 Install New Software... 选项,进行更新,如图 1.7 所示,在 Work with 输入框中输入"https://dl-ssl.google.com/android/eclipse/",单击 Add 按钮,如果是第一次添加上述地址,会弹出输入名称窗口,随便填写一个即可,比如"adt"。

当 Eclipse 查询到更新内容时,会自动列出,选择需要安装的插件(建议全选),如图 1.8 所示,单击 Next 按钮进行下载。这将会花费很长时间。

当漫长的下载工作完成后,会弹出安装窗口,如图 1.9 所示,这表明 Eclipse 所需要的插件已经全部下载完毕,此时单击 Next 按钮进行安装工作。勾选 I accept…选项,开始安装。安装过程如出现提示,单击 Yes 或 OK 按钮,继续安装就可以。当安装完成后,会提示 Eclipse 需要重启(注意是 Eclipse 软件需要重启,不是计算机需要重启),单击 Restart now 即可。

重新启动后的 Eclipse 在工具栏上会新增加一个图标 (提示:opens the Android SDK and AVD manager)。选择 Window→Preferences 选项,打开 Preferences 窗口,选择左边的 Android,在 SDK Location 文本框中输入 SDK 的根目录,如"E:\Android\android-sdk",如图 1.10 所示。该开发环境是否安装成功,后面会利用第 2 章的例子来验证(迫不及待想一试身手的读者可以直接前往第 2 章)。

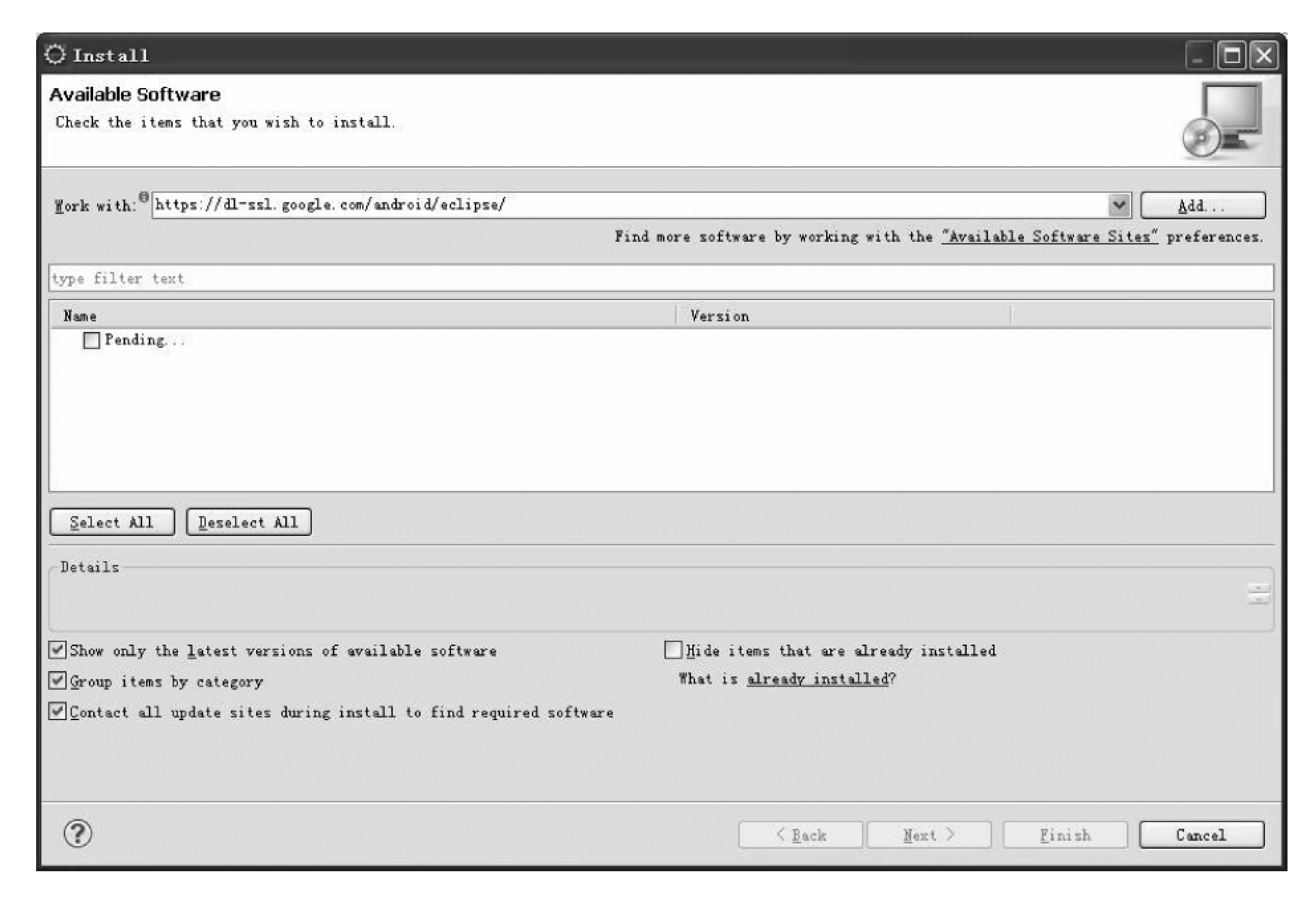


图 1.7　Eclipse 软件更新界面

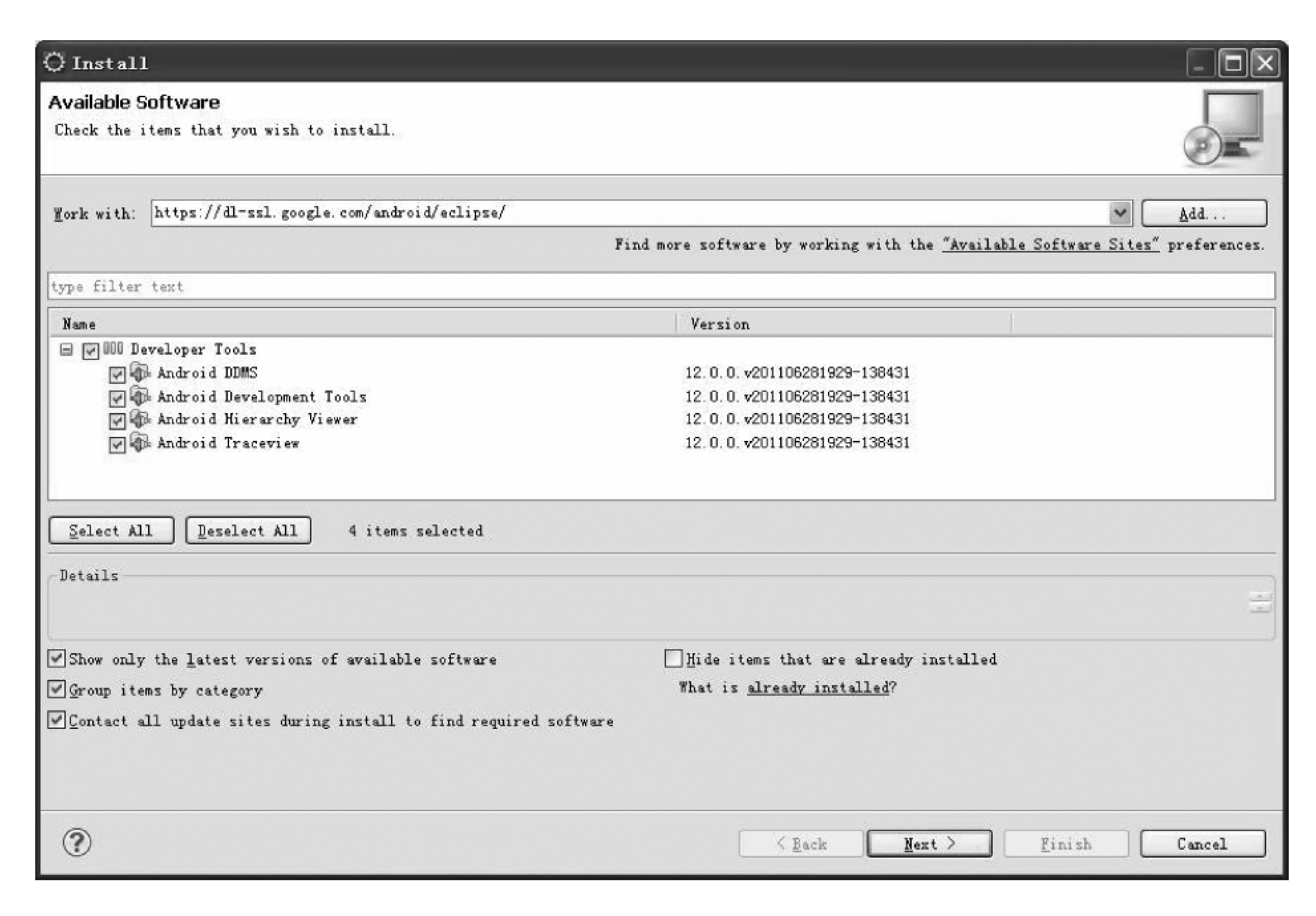


图 1.8　Eclipse 搜索更新内容

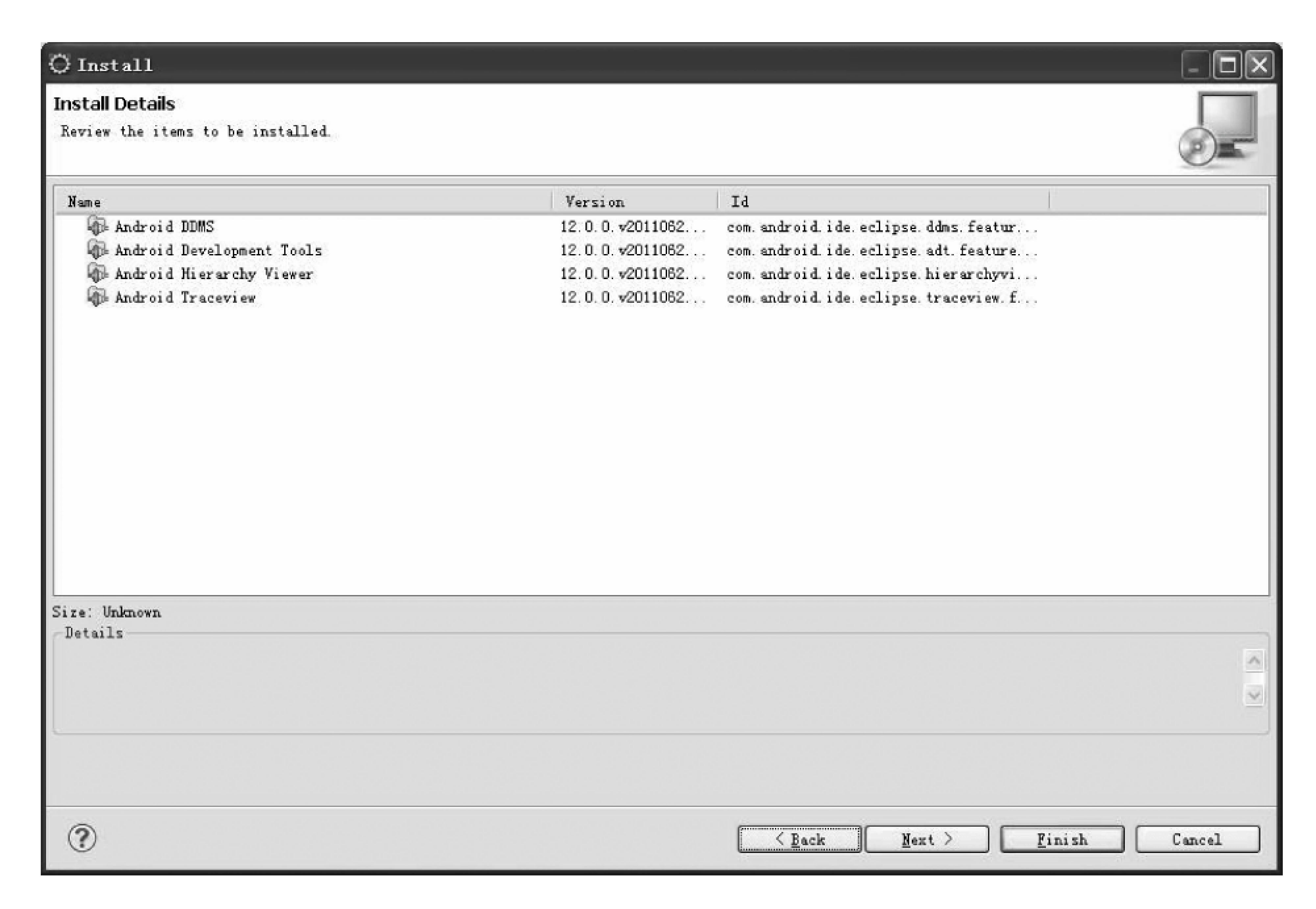


图 1.9　Eclipse 完成下载插件

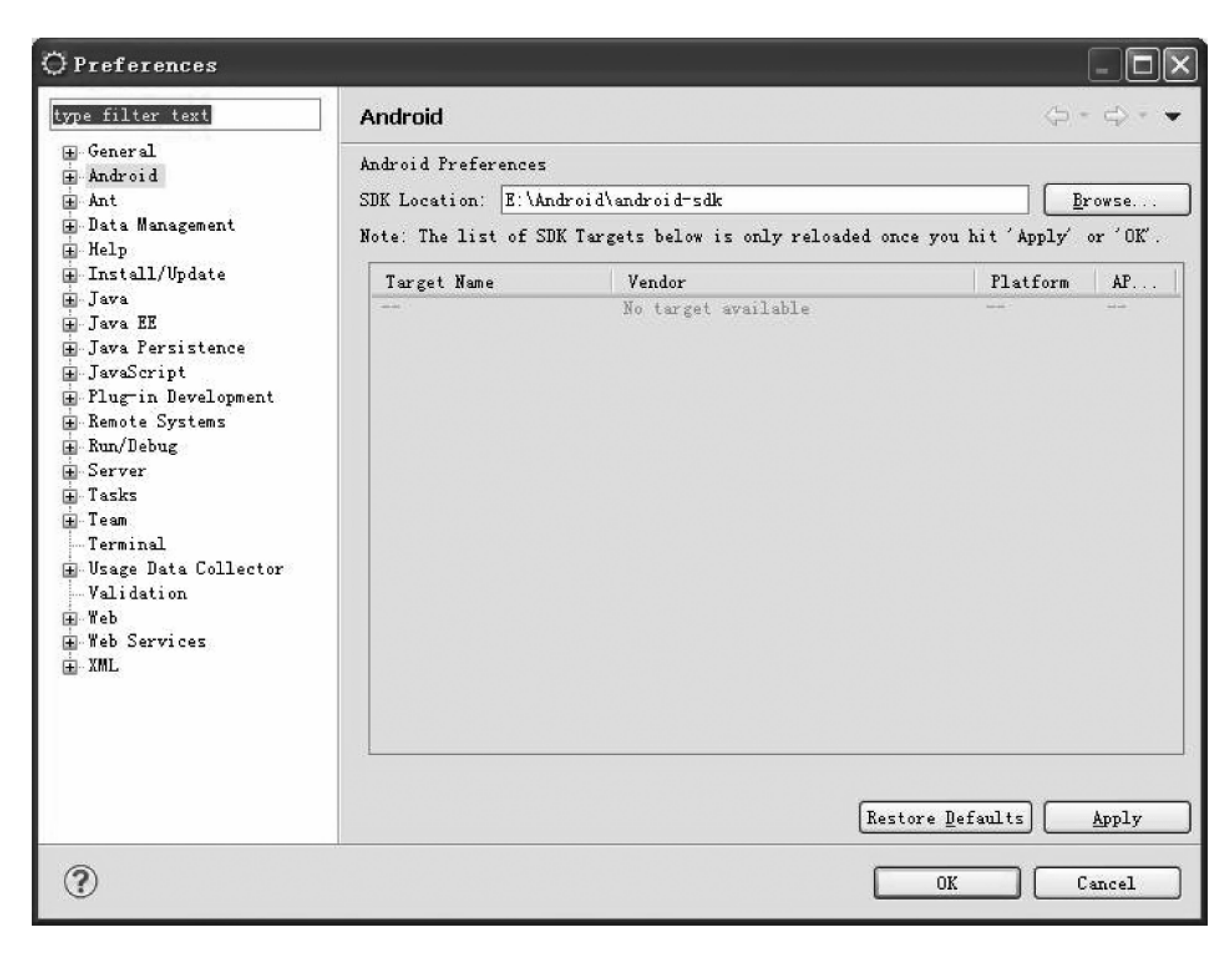


图 1.10　配置 SDK Location

1.3 Java 数据类型与运算符

1.3.1 基本数据类型

学习过编程语言的读者都知道，任何一门编程语言都有基本数据类型，这就像每门功夫都有自己的招式一样。这些基本类型是不可以拆解的，相当于基本单位。在 Java 中基本的数据类型共有 8 种，详见表 1.3。

表 1.3 Java 基本数据类型

类型	关键字	描　述	大小/格式
整型	byte	字节型($-2^7 \sim 2^7-1$)	1 个字节
	short	短整型($-2^{15} \sim 2^{15}-1$)	2 个字节
	int	整型($-2^{31} \sim 2^{31}-1$)	4 个字节
	long	长整型($-2^{63} \sim 2^{63}-1$)	8 个字节
浮点型	float	单精度浮点型(IEEE 754—1985 标准)	4 个字节
	double	双精度浮点型(IEEE 754—1985 标准)	8 个字节
字符型	char	单个字符($0 \sim 2^{16}-1$)	2 个字节
布尔型	boolean	布尔型数值(true 或 false)	1 位

1. 整型——byte、short、int 和 long

整型常量可用十进制、八进制或十六进制形式表示，以 1～9 开头的数为十进制数，以 0 开头的数为八进制数，以 0x 开头的数为十六进制数(对于以 0 开头的小数，因为属于浮点数，所以不在此列)。Java 中所有的整型量都是有符号数(signed)。如果想表示一个长整型常量，需要在数后明确地写出字母 L 或 l(一般使用大写 L，这样容易辨识)。

2. 浮点型——float 和 double

Java 浮点类型遵从标准的浮点规则，用 Java 编写的程序可运行在任何机器上。如果数值常量中包含小数点、指数部分(字符 e)，或其后跟有 F(f)或 D(d)，则为浮点数。浮点的指数形式使用科学记数法(e 形式)，e 前面的数字可以是一个带有小数点的数字，也可以是一个不带小数点的数字。e 后面的数字中不能包含小数点。例如，8.65 * 10^8 写成 8.65e8，8.65 * 10^-8 写成 8.65e-8。

3. 字符型——char

单个字符用 char 类型表示。一个 char 表示一个 Unicode 字符，其值用 16 位无符号整数表示，范围是 0～65 535。实际上，Java 源代码使用的是 Unicode 码，而不是 ASCII 码。Unicode 码用 16 位表示一个字符，因此，Unicode 字符集中的字符数可达 65 535 个，比通常使用的 ASCII 码字符集大得多。Unicode 兼容了许多不同的字母表，包括常见语种的字母。英文字母、数字和标点符号在 Unicode 和 ASCII 字符集中有相同的值。char 类型的常量值必须使用一对单引号括起来。

4. 布尔型——boolean

boolean 类型只有两个取值：true 和 false，它们全是小写形式。Java 不允许数值类型和布尔类型之间进行转换。

1.3.2 运算符

Java 中的运算符主要包括算术运算符、关系运算符、逻辑运算符、条件运算符等。

1. 赋值运算符

赋值运算符是一个等号=，比如 A = B，那么 B 处的内容就复制到 A。由于 Java 语言是强类型的语言，所以赋值时要求类型必须匹配，如果类型不匹配需要能自动转换为对应的类型，否则将报语法错误。

2. 算术运算符

Java 的算术运算符有加号（+）、减号（-）、乘号（*）、除号（/）以及模数（%，从整数除法中获得余数）。整数除法会直接去掉小数，而不是进位。

3. 自动递增、递减

对于前递增和前递减（如++A 或--A），会先执行加 1 或减 1 运算，再生成值。对于后递增和后递减（如 A++或 A--），会先生成值，再执行加 1 或减 1 运算。

4. 关系运算符

关系运算符包括<、>、<=、>=、= =、! =，关系运算产生的结果是布尔值。若关系表达式的关系是真实的，则运算结果为 true（真）；若关系不真实，则结果为 false（假）。关系运算符经常用在 if 控制语句和各种循环语句的表达式中。

Java 中的任何类型，包括整数、浮点数、字符，以及布尔型都可用“==”来比较是否相等，用“! =”来测试是否不等。注意 Java（C 和 C++一样）比较是否相等的运算符是两个等号，而不是一个等号。

5. 逻辑运算符

逻辑运算符包括与（&&）、或（||）、非（!），逻辑运算的结果也是布尔值。&& 作为逻辑运算符与使用，也被称作短路与，运算时先判断符号前面的表达式的值，如果能够确定整个表达式的值，则不进行符号后面的表达式的运算。同理，||也被称作短路或。另外，&和|可作为位运算符使用。

6. 条件运算符

对于布尔表达式“? 表达式 1：表达式 2”，如果布尔表达式的结果为 true，就计算并返回表达式 1 的值，否则计算并返回表达式 2 的值。

有数据类型，有运算符就可以组建表达式了，在表达式中，运算符的先后顺序如表 1.4

所示。

表 1.4 运算符及优先级

操作符类型	操 作 符
括号	( )、[ ]
一元操作符	+、-、++、--
算术操作符	*、/、%、+、-
关系操作符	>、<、>=、<=、==、! =
逻辑操作符、位操作符	&&、||、&、|、^
条件操作符	A > B ? X : Y
赋值操作符	=、*=、/=、+=、-=

1.3.3 不同数据类型间的转换

整型、实型、字符型数据可以混合运算。运算时，不同类型的数据先要转换为同一类型然后进行运算。对于表示数字的几种数据类型，在下面序列中可以将任意类型的值赋给沿着箭头方向出现在其后面的任意类型的变量：

```
byte→shor→int→long→float→double
```

上述数据类型的转换方式可以自动完成，而不需要显式说明，这种数据转换方式一般称为自动类型转换。但当表示位数多的类型向表示位数少的类型转换时，需要显式说明，这种数据类型转换方式一般称为强制类型转换。例如：

```
int a = 360;
byte b;
b = (byte) a;
```

1.3.4 引用数据类型

上述基本数据类型，几乎任何一种编程语言都具备，在 Java 中，除了上述基本数据类型外，还支持引用数据类型。引用数据类型是由基本数据类型组合而成的，是比较复杂的数据。类型相同的一组数据构成数组，不同类型且有内在联系的一组数据构成类和接口。如同 C 语言中的数组和结构体。在软件开发过程中，使用引用数据类型是比较频繁的，因为这样的数据是有意义存在的。为什么称为引用类型，在学习完类和对象后，可以自己思考，现在姑且记住即可。

1.4 Java 基本流程控制语句

流程控制提供了控制程序步骤的基本手段，是程序的核心部分。Java 的控制流程用于使程序按正确顺序逐步执行，为程序提供了执行方向。基本控制语言与 C 语言的控制语句一样，主要有分支语句、循环控制语句和辅助语句。

1.4.1　分支控制语句

分支控制语句提供几个程序分支，根据某个判定条件，选择执行其中一条或几条语句。常用控制语句是 if 和 switch，以及它们的变化形式，如下：

```
if(判定条件){
    语句块;
}
```

当判定条件成立时，执行语句块，否则不执行。

```
if(判定条件){
    语句块 1;
}else{
    语句块 2;
}
```

当判定条件成立时，执行语句块 1，否则执行语句块 2。

```
if(判定条件 1){
    语句块 1;
}else if(判定条件 2){
    语句块 2;
}else if(判定条件 3){
    语句块 3;
}…
```

当判定条件 1 成立时，执行语句块 1，否则检查判定条件 2，如果成立，执行语句块 2，否则检查判断条件 3，如果成立，执行语句块 3，……

```
switch(表达式){
case 结果 1 : 语句块 1; break;
case 结果 2 : 语句块 2; break;
case 结果 3 : 语句块 3; break;
⋮
default : 语句块 n;
}
```

根据表达式的运算结果，选择执行 case 分支。需要注意的是：

- 控制语句可以嵌套，在嵌套时，内层语句要完全被外层语句包裹，不允许内外两层出现交叉。
- 判定 if 语句中的判定条件必须是明确的 true 或 false，不能使用 0 代表 false，非 0 代表 true，这一点与 C 语言不同，switch 语句中的表达式必须能够明确地得出与 int 兼容的值。
- case 语句中的结果必须是明确的已知的常量。
- case 语句块后面需要使用 break，否则该 case 后面的其他 case 也会执行，直到遇见 break 或结束。
- default 语句是可选项，在 case 结果无一匹配的情况下，会默认执行 default 分支。

1.4.2 循环控制语句

根据判定条件，循环控制语句决定是否执行循环体中的语句块。可以实现循环的常用控制语句有 while、do…while、for，以及它们的变化形式。如下：

```
while(判定条件){
    语句块;
}
```

如果判定条件成立，则执行语句块，然后继续判定条件是否成立，直到判定条件不成立了，即结果为 false，循环结束。

```
do{
    语句块;
}while(判定条件);
```

该循环会先执行一次语句块，然后检查判定条件是否成立，如果成立，继续执行循环，直到判定条件不成立。

```
for(循环变量初始化; 判定条件; 循环变量的改变){
    语句块;
}
```

先进行变量的初始化，在循环过程中，该部分只会被执行一次，然后检查判定条件是否成立，如果成立，执行语句块，接着转到变量的改变，当变量改变后，继续检查判定条件，成立则执行语句块，直到检测判断条件不成立。

```
for(元素类型 名称: 集合名称){
    语句块;
}
```

该循环被称为增强 for 循环，可以很容易遍历一个集合中的所有元素，这种用法是 JDK 1.5 之后出现的。关于循环控制还需要注意以下几条。

- 循环语句可以相互嵌套，在循环嵌套时不要出现语句块交叉，且嵌套的层次不宜过多，如果要三次以上的嵌套才能完成，则需要考虑是否可以采用其他方式实现。
- 循环语句可以有相应的形式变化，与 C 语言的循环控制一样。
- 循环语句和分支语句有机结合可以实现比较复杂的代码。
- 循环语句常配合辅助语句使用，在适当的时候结束本次循环或直接跳出循环。

1.4.3 辅助语句

这里所指的辅助控制语句是指 break 和 continue。switch 选择流程中使用过 break 语句，除了用于退出 break 流程，使程序从 switch 语句后的第一条语句开始执行外，break 还用于退出 while、do…while 及 for 等循环流程。continue 语句用于提前结束本次循环，跳过循环体中下面没有执行的语句，直接进行终止条件判断，来决定下一次循环是否继续执行。

1.4.4　其他控制语句

以上介绍了常用的流程控制语句，除了在真/假的表示上，其他与 C 语言中的程序控制几乎完全一致。Java 语言还可以使用其他的方式改变程序的流程，如果程序在执行过程中出现异常，则可能会打破原先的预定顺序，改用紧急处理方式。这就好比要举办一项活动，正常情况下，肯定是按部就班，从头到尾，按计划执行，如果中途出现意外情况，如地震，就要执行紧急情况处理方案，甚至终止整个活动。在这一点上，可以看出 Java 的编程很有人情味，在后面的学习中，读者会慢慢体会，编程几乎是在模仿社会的各种活动。

Java 中提供了处理这种异常的语句，如下：

```
try{
    语句块;
}catch(异常 1){
    异常处理;
}catch(异常 2){
    异常处理;
}…
finally{
    语句块;
}
```

异常控制语句虽然可以起到控制流程的作用，但这不是 Java 提供该语句的初衷，异常处理语句的主要作用是防止出现异常，避免引起程序崩溃。如果出现异常，按照预定方案捕获，并进行相应处理，最后执行 finally 中的语句块。关于异常的处理在后续章节中会详细介绍。

1.5　Java 语言的特点

Java 语言是在网络应用大繁荣前出现的，在促进网络应用多样化的同时，不断成熟完善，具备了自己独有的特点。

1.5.1　强类型

在编程过程中，一个变量的数据类型必须明确定义，且给该变量的赋值必须与定义类型匹配。这不同于一些编程语言采用 var 定义一个变量，对值的类型不做规定。强类型虽然给编程带来了麻烦（必须明确指明变量类型），但是产生的好处却是显而易见的，首先类型被明确以后，减少了数据转换可能产生的异常或错误的发生；其次，代码在编译时的速度得到提升。

1.5.2　完全面向对象

面向对象是目前主流编程语言都一致遵循的编程思想，它是从现实世界中客观存在的事物出发，构造软件系统，并在系统构造中尽可能运用人类的自然思维方式，以现实世界中的事物为中心来思考问题、认识问题，并根据这些事物的本质特点，把它们抽象地表示为系统中的对象，作为系统的基本构成单位。

换句话说，面向对象就是一切都是围绕着对象进行软件的设计与开发，而这个对象就是从现实世界中映射到程序中的一个概念。比如现实世界中有个学生叫小张(不是单纯名字，而是整个活生生的人)，他会有自己的一些特征(称为属性)，体重 60kg，身高 180cm 等，也会有自己的功能和行为(称为方法——能做什么)、弹钢琴、选课、考试等。这个名叫小张的学生只是众多学生中的一员，是学生这个概念的其中一个实体。转换到程序设计中，例如开发一套学生管理系统，肯定会涉及很多学生，那么这个学生就称为类(class，一组具有共同属性的对象的抽象，或者说概括、描述等)的概念，那么名叫小张的学生就成为学生这个类的一个实例(Instance，一个抽象概念的具体实现)，名字小张，只是这个具体实例的引用(Reference，指向一个实例，与 C 语言中的指针类似)。

在上面的叙述过程中一直没有出现对象(Object)这个概念，不同的书上对其说法不一，其实读者只要明白什么是实例，什么是引用，以及它们的关系，就可以了，无须对这些概念纠缠不清。现在澄清一下，严格地说对象就是实例，通俗地说对象就是实例和引用。所以，在面向对象编程中，提到比较多的是类(class)和对象(Object)这两个概念，类就是具有共同属性的一组对象的抽象，反过来，对象就是对一个类的具体实现。

采用面向对象的编程思想，在设计与开发软件时，首先考虑的不是先干什么，后干什么，而是要考虑完成这个软件的功能，需要哪些对象参与，这些对象都应该具有哪些属性和方法，它们该如何配合等。举个例子，将一面白墙刷出蓝色，这就是一项工程，要完成此工程应该由哪些对象参与呢(读者可以仔细思考)？首先，参与的对象应该有墙壁、刷子、粉刷工、蓝色涂料，或许还需要梯子等。其次，这些对象都有哪些属性和方法呢？简单来说，墙壁肯定有宽度和高度的属性，粉刷工肯定会刷墙的方法。如果在软件的设计与开发中，读者能这样考虑事情，那么恭喜您已经在采用面向对象的思想想问题了。

当然，面向对象不是三言两语就能弄清的，否则也不能发展为一种技术了。Java 就是纯粹地采用这种面向对象思想，如果读者对面向对象感兴趣，学习 Java 语言肯定是不错的选择。

1.5.3 多线程

Java 编写的程序都是运行在 Java 虚拟机(JVM，这也是 Java 可以跨平台的原因)中，在 JVM 的内部，程序的多任务是通过线程(线程是不同于进程的)来实现的。每用 Java 命令启动一个 Java 应用程序，就会启动一个 JVM 进程。

一般常见的 Java 应用程序都是单线程的。例如，用 Java 命令运行一个最简单的 Java 应用程序时(如代码 01-2)，就启动了一个 JVM 进程，JVM 找到程序的入口点 main()，然后运行 main()方法，这样就产生了一个线程，这个线程称为主线程。当 main 方法结束后，主线程运行完成。JVM 进程也随即退出。

在 Java 编程中，提供了两种方法可以实现多线程：继承 Thread 类、实现 Runnable 接口，操作起来比较容易，在后续的介绍中会逐渐向读者说明。

1.5.4 可移植性

Java 号称 Write once，run anywhere，也就是这里要讲的可移植性，主要的原因就在于 Java 采用了 Java 虚拟机(JVM)。在前面的介绍中，曾采用 javac 命令，将编写好的代码编译

成.class 文件(一种中间码,并非二进制代码),然后通过 java 命令来运行它。那么 Java 为何要如此多费周折呢?

如果 Java 直接编译成系统能识别的二进制码,很可能出现这样的问题:在 Windows 下编译是 1100,而 Linux 下是 1001,这样 Java 在 Windows 下编译后无法在 Linux 下运行。所以 Java 干脆不编译成二进制文件,而是先编译成字节码(中间码),由 JVM(Java 虚拟机)来解释执行,而这个 JVM 对于主流的操作系统都有相应的版本(可以到官方网站下载各操作平台的 JVM),目的就在于将统一的中间码编译成对应操作系统识别的二进制码,然后执行。所以,Java 代码的编写与操作系统无关,最终只会被编译成.class 文件,在 Windows 中需要由 Windows 版本的 JVM 来执行,要是到了 Linux 下,只要下载 Linux 版本的 JVM 来执行就可以了。这样就实现了 Java 的跨平台、可移植性。

1.5.5 其他特点

除了上面介绍的 Java 语言的特点外,Java 语言还具备以下特点。

(1) 健壮性,使得程序在运行过程中不易崩溃,前面介绍的 try…catch 异常捕获机制就是其中的一个表现。

(2) 安全性,Java 非常强调安全性,以确保建立无病毒且不会被侵入的系统。

(3) 扩展性,Java 是一种比 C 或 C++ 更具动态特性的语言,它在设计上强调为不断发展的运算环境提供支持,提供了很多机制,方便对软件进行扩展,而不影响原有功能。

习 题

1. 编写、运行 Java 程序需要经过哪些主要的步骤?
2. Java 语言中基本数据类型有哪些?
3. 如何理解类、接口、引用、对象? 它们之间有什么关系? 有什么区别?
4. 抽象类与接口有哪些相同点? 有哪些不同点?
5. 发生方法重载的条件是什么?
6. 重载与覆盖(复写)有何区别?
7. Java 中访问修饰符有哪些? 它们所限制的范围是什么样的?
8. 如何理解字符串中的字符串与对象类型的字符串?
9. 执行以下程序,输出的结果如何?

```
public static void main(String[] args) {
    int a,b;
    boolean c,d;
    a = b = 5;
    c = a++> 5&&++b> 5;
    d = a++> 5||b++> 5;
    System.out.println(a + " " + b + " " + c + " " + d );
}
```

10. 熟练掌握 J2SE 中集合框架、多线程、I/O 流、JDBC、网络编程的相关知识。



CHAPTER 2

第2章

初识Android平台

本章主要内容：

Android 平台介绍；
Android 模拟器；
Android 第一个项目；
初识 Activity。

2.1 Android 平台介绍

2.1.1 Android 平台的由来

Android 是一款移动设备的操作系统，被广泛地用于智能手机、平板电脑、智能电视等终端设备。由 Android 公司设计开发，并以公司命名，该系统由操作系统、中间件、用户界面和应用程序组成，整体架构主要分为三个部分。底层基于 Linux 内核，采用 C 语言开发，提供基本功能；中间层包括调用函数库和运行虚拟机，采用 C++ 开发；最上层（与 APP 开发者最为接近的一层）包括各种应用程序、如通话程序、短信程序等，可以自由开发，采用 Java 开发语言；整个系统具有自由开放的特征，不存在阻碍移动产业创新的专有权障碍。2005 年由 Google 公司收购注资，随后又成立了开放手机联盟，Android 系统的功能得到了进一步的完善，Google 公司通过与相关软硬件开发商、设备制造商、运营商等企业结成合作伙伴，在移动产业内建立了开放式环境。图 2.1 是 Android 图标，几乎是家喻户晓。



图 2.1　Android 图标

鉴于其开源的特性，该系统一经推出便受到众多终端制作企业的青睐，纷纷采用 Android 作为手机操作系统。2011 年该系统市场占有率便跃居全球第一，其市场占有份额一度超过全球智能手机操作系统的一半以上。台湾宏达国际电子（HTC）、摩托罗拉、韩国三星电子、LG 电子和中国移动等都是开发手机联盟的成员，阵容可谓庞大。随着 Android 操作系统的普及，该平台的应用软件越来越受欢迎，应用程序的开发速度有待进一步提高，

Android 应用程序开发者的需求量也越来越大，伴随着巨大的产业需求，国内 Android 系统开发人才越来越多，Android 应用开发及系统开发的工程师将成为未来几年最为热门的职业之一。

2.1.2　Android 历届版本

Android 平台自 2007 年 11 月发布首款商业操作系统(beta 版)开始，不断更新、完善该平台，陆续发布了多个版本，这些版本的命名是按照大写字母表的顺序来的，截至目前，最新版本是 Android 4.1 果冻豆 Jelly Bean(字母已经排到"J")，本节对 Google 发布的这些版本以及功能进行简要说明。

(1) Android 1.5，代号 Cupcake，于 2009 年 5 月发布，这是第一个主要版本，用户操作界面得到极大改善，开始吸引开发者的目光，该版本主要完善的功能有：

- 拍摄和播放视频，并可以上传到 YouTube；
- 支持立体蓝牙耳机；
- 采用 WebKit 技术实现浏览器，支持复制、粘贴和在页面中进行搜索；
- 提高 GPS 性能；
- 提供屏幕虚拟键盘。

(2) Android 1.6，代号 Donut，于 2009 年 9 月发布。搭载该操作系统的 HTC Hero 智能手机获得了意想不到的成功，Android 开始吸引更多人的目光，包括竞争者苹果和微软。该版本主要完善的功能有：

- 重新设计 Android Market；
- 增加手势支持；
- 支持 CDMA 网络；
- 增加文本转语音的功能；
- 支持虚拟个人网络(VPN)；
- 支持更多的屏幕分辨率。

(3) Android 2.0/2.1，代号 Éclair，于 2009 年 10 月发布。这是继 Android 1.5 之后的又一个主要版本，主要更新有：

- 优化硬件速度；
- 用户界面的改良；
- 浏览器支持 HTML 5；
- 支持蓝牙 2.1；
- 支持动态桌面设计；
- 改进 Google Map。

(4) Android 2.2，代号 Froyo，于 2010 年 5 月发布，主要更新有：

- 支持将软件安装到扩展内存；
- 增加 USB 分享器和 Wi-Fi 热点功能；
- 浏览器集成 Chrome 的 V8 JavaScript 引擎。

(5) Android 2.3，代号 Gingerbread，于 2010 年 12 月发布，主要更新有：

- 支持更大屏幕尺寸和分辨率；

- 系统级的复制和粘贴；
- 重新设计多点触屏键盘；
- 优化游戏开发支持；
- 支持更多的传感器。

(6) Android 3.x,代号 Honeycomb,于 2011 年 2 月发布,该版本开始支持平板电脑,主要更新有:

- 优化针对平板电脑的功能；
- 全面支持 Google Map；
- 3D 加速处理和支持多核心处理器；
- 支持操作杆和游戏控制器。

(7) Android 4.0,代号 Icecream Sandwich,于 2011 年 4 月发布,该版本主要的更新内容有:

- 统一手机和平板电脑操作系统,应用可以根据设备选择最佳显示方式；
- 提升硬件的性能以及系统的优化,提示系统运行的流畅度；
- 脸部识别进行锁屏；
- 全新的 3D 驱动,游戏支持能力提升；
- 支持 Wi-Fi 直连功能。

(8) Android 4.2,代号 Jelly Bean,于 2012 年 10 月发布,比较重要的改进和升级体现在:

- Photo Sphere 全景拍照；
- 键盘手势输入；
- Miracast 无线显示共享；
- 手势放大缩小屏幕；
- 为盲人用户设计的语音输出和手势模式导航功能。

(9) Android 4.3,代号 Jelly Bean,于 2013 年 7 月发布,比较重要的改进和升级体现在:

- 提升图形性能,增加硬件加速 2D 渲染优化了流绘图命令；
- 支持 OpenGL ES 3.0。

Google 公司自 2009 年以来,不断推出新的产品,增加完善了 Android 平台的功能,至于各个版本的新增功能,读者可以在选择开发平台时,应该本着尽量多的人都可以使用的原则。在后期的学习开发过程中,主要采用 Android 2.3.3 和 Android 4.2 这两个版本。

2.1.3 Android 平台的特征

Android 被称作移动设备打造的首个真正完整和开放的移动平台,该平台设计之初便考虑到为应用程序开发者提供二次开发的可能性,具有健壮的应用程序框架,提供丰富的应用程序开发接口。Android 以开放代码为前提,应用程序开发者可以比较自由地获取访问硬件设备的权限,在这个平台上开发应用程序不需要任何许可证和版权费用,这也是能吸引众多开发者参与的因素之一。

1. 免费与开放

Android 是一个完全开放源代码的平台，无论应用程序开发者还是手机制造商，都具有自由的开发空间。这种优势可以吸引众多的开发者和手机制作商加入到该平台的联盟阵营，壮大 Android 平台的影响力，积累人气，丰富应用程序，从而能够赢得更多的消费人群。

应用程序开发者可以任意发布应用程序，既可以编写该平台的免费软件，也可以开发需要授权的软件，获得一定的报酬。系统开发者需要遵循 GPL v2 协议，任何改进必须遵循开放源代码的协议约定。这种开放性在方便开发者的同时，也会导致该平台会出现众多版本不统一的情况，Google 公司有可能会调整相应的开放原则。

2. 开发语言与工具

Android 平台的应用程序可以采用 Java 语言进行编写，这给众多熟悉 Java 语言的开发者提供了更为广阔的发展空间，在短时间内为 Android 应用程序的开发找到了强有力的支持。Java 语言以开源和开放为特色，具有大量的代码模型可以参考。Android 应用程序的集成开发环境目前较为流行，可以免费获得 Eclipse（需要安装 ADT 插件），该软件具有众多的开源社区和资源可以免费试用，也能够简化代码开发的复杂度。

3. 整合 Google 应用与服务

Android 作为 Google 重磅推出的产品，承担了 Google 公司太多的使命。Android 无缝结合了 Google 很多优秀的应用与服务，比如搜索、天气预报、GoogleTalk、地图、Gmail 等。这使得 Android 拥有其他系统无可比拟的优势。用户在使用 Android 在线服务时，可以与计算机上使用的 Google 服务进行完全整合，实现 Google 服务的完全同步。

4. 分层的体系结构

Android 平台采用分层的架构，如图 2.2 所示，从上到下分别是应用程序层、应用程序框架层、运行环境与运行库层、Linux 内核层。其中，应用程序层和应用程序框架层是 Java 语言编写的程序，运行环境与库层是由 C/C++ 编写，其中虚拟机部分可以运行 Java 语言，Linux 内核层是 Android 的最底层，主要由各种驱动程序组成。

作为应用程序开发者，将来开发的应用程序应位于应用程序层。该层上包括主屏幕、SMS 短消息程序、联系人管理器、浏览器、日历、地图等一系列已经实现的应用程序（在 2.2 节介绍的 Android 模拟器中可以了解这样的应用程序）。所有的应用程序都是使用 Java 语言编写。

应用程序框架层为应用程序层的开发者提供 APIs，实际上是一个应用程序的框架。由于上层的应用程序是采用 Java 编写的，这些框架可以提供 Java 语言直接调用包含 UI 程序中所需要的各种控件以及服务，常见的有：

- View System，包括列表（list）、网格（grid）、文本框（textbox）、按钮（button）等控件。
- Content Providers，一个应用程序可以访问另一个应用程序的数据（如联系人数据库），也可以共享自己的数据。
- Resource Manager，资源访问的管理，如字符串、图形和布局文件的访问。

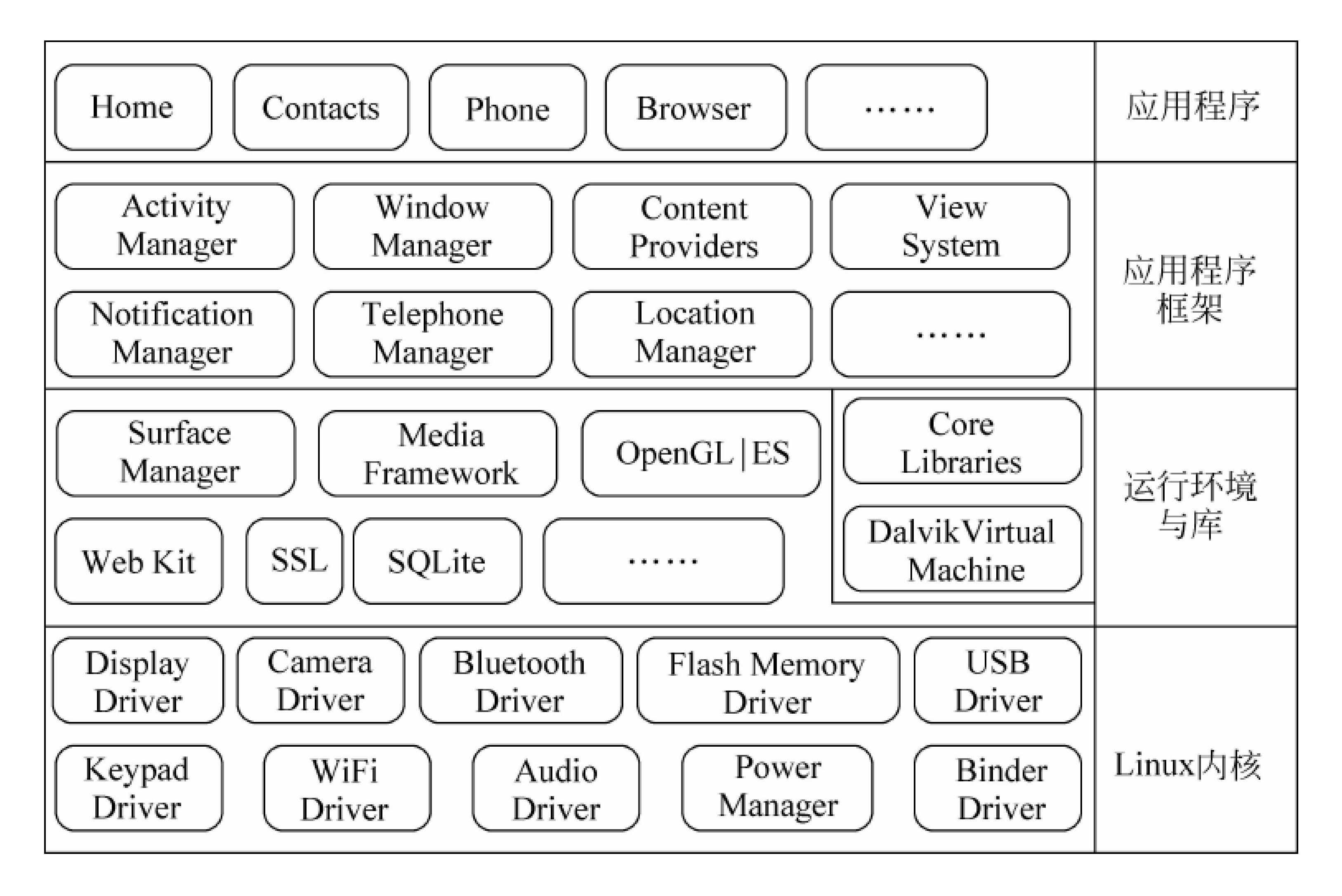


图 2.2 Android 平台体系结构

- Notification Manager，应用程序可以在状态栏（手机屏幕的最上方）中显示自定义的提示信息。
- Activity Manager，管理应用程序生命周期并提供常用的导航回退功能。

第三层包括运行环境和程序库。Android 应用程序在独立的进程中运行，拥有独立的 Dalvik 虚拟机。Dalvik 可以同时高效地运行多个虚拟系统。程序库包含一些 C/C++库，可以被 Android 系统中不同的组件使用，通过 Android 应用程序框架为应用程序开发者提供各种服务。

体系结构的最底层是 Linux 内核层，Android 平台的核心系统服务依赖于 Linux 操作系统的内核，如安全性、内存管理、进程管理、网络协议和驱动模型等，这也是硬件和软件之间的抽象层。

5. 众多 App Market

Android 平台为开发者提供了开放的开发环境，开发者可以自由开发免费软件、共享软件、试用软件，也可以开发依靠植入广告或直接付费的盈利软件。如果想达成这种目标，必须将应用程序推送到用户端才可以，App Market 就是服务开发者和用户的一个平台，它允许开发者发布应用程序（不同平台有不同的审查制度），Android 用户也可以自由下载应用程序到自己的移动设备。作为一个连接开发者和用户的平台，应用程序商店具有很大的发展空间、诱人的利润，是众多运营商、手机制作商、软件开发商的必争之地。短短两年时间，国内的应用程序商店就已达数十个。Android Market 是 Google 官方应用程序商店，面向全球用户，在国内人气比较旺的应用程序商店有机锋市场、安卓市场、安智市场、腾讯应用中心、网易应用、AppChina 应用汇和 360 宝盒等。图 2.3 是机锋网开发者社区应用程序管理界面。

此外，这些应用程序商店都有与之匹配的开发社区，如果想快速成为 Android 开发高手，加入 Android 开发者社区是不错的选择，在那里可以获取前沿知识、学习资料、免费资

源，同时也可以向行业内高手请教。图 2.4 是 eoe 移动开发者社区 Android 板块的首页，对于众多国内开发者，选择成熟度较高的国内社区网站可以更快、更便捷地掌握 Android 开发知识。



图 2.3 机锋网应用程序发布平台



图 2.4 eoe Android 开发者社区

以上介绍可以帮助读者了解 Android 平台的发展轨迹，熟悉 Android 平台目前的市场信息，明确学习方法，对 Android 平台的发展前景有一个很好的认识。今后在学习过程中，还需要注意的事项有：

- Android 平台应用程序开发主要采用 Java 语言，所以要熟练掌握 Java 语言的使用。
- 学会学习，目前有很多开源社区都提供丰富的资源，可以免费获取，这些都是学习 Android 应用开发的捷径。
- 学习任务任重而道远，需要循序渐进，切不可急于求成。

2.2 Android 应用程序的测试设备

进行 Android 应用程序开发，需要有测试运行的环境，模拟器就是 Android SDK 自带的一个移动设备模拟软件（PC 上的虚拟智能电话）。借助模拟器可以不使用物理设备（真

机)即可预览、开发和测试 Android 应用程序。如果配有 Android 操作系统的手机,也可以通过连接 USB 线,选中真机测试运行。

2.2.1 虚拟机的创建

创建虚拟机(模拟器)有多种方式,既可以在 Android SDK 目录下直接运行 AVD Manager.exe,也可以在 Eclipse 中选择 Window→AVD Manager 命令,或者直接单击 (opens the Android Virtual Device Manager)图标,打开 Android Virtual Device Manager 窗口,如图 2.5 所示。

图 2.5 Android 虚拟机管理窗口

列表中是曾经创建的虚拟机,选中后可以激活 Edit、Delete、Repair、Details 操作,通过右侧的 New 按钮创建新的虚拟机,打开虚拟机创建对话框,如图 2.6 所示。

虚拟机在创建过程中有 4 个参数需要设置(Name、Target、SD Card、Skin),会影响到虚拟机的使用。Name 参数是虚拟机的名称,可以自己设置;Target 是模拟器操作系统的版本,Android 平台的不同版本具有不同的特性和功能,可以查看前面关于 Android 平台历届版本的说明;CPU/ABI 由选择的目标操作系统版本决定,无法自行设置;SD Card 是虚拟机的 SD 卡设置,可以新建一个 SD 卡文件,直接输入 SD 卡的存储大小即可(单位 MiB),也可以选择已经存在的 SD 卡文件;Snapshot 为虚拟机的快照功能,虚拟机在启动时比较耗时,具有快照功能后可以比较快地启动,这个参数目前可以不设定(有不同版本可能会引起启动错误);Skin 有两种设定方式,Built-in 是 Android 各版本直接支持的屏幕,Resolution 是用户自定义的屏幕,此处建议选择 HVGA(这涉及屏幕密度、分辨率,后续内容有介绍);Hardware 可以选择该模拟器的硬件支持,目前先不做设置。全部参数设定完后,就可以单击 Create AVD 按钮创建虚拟机,关键参数如图 2.7 所示。

已经创建成功的虚拟机可以通过虚拟机管理窗口,选择 start 启动,在 Launch Options 窗口直接选择 Launch(没有做快照,直接启动即可),开始启动虚拟机。启动过程比较慢,最终效果如图 2.8 所示。

虚拟机主要分为左边的模拟手机界面和右边的模拟手机键盘的按键。在手机界面中可以像真正的手机一样进行操作,其中内置了很多 Google 公司的应用程序,如拨打电话、发短信、时钟、浏览器、Google Map 等。在使用虚拟机时应该注意以下两点。

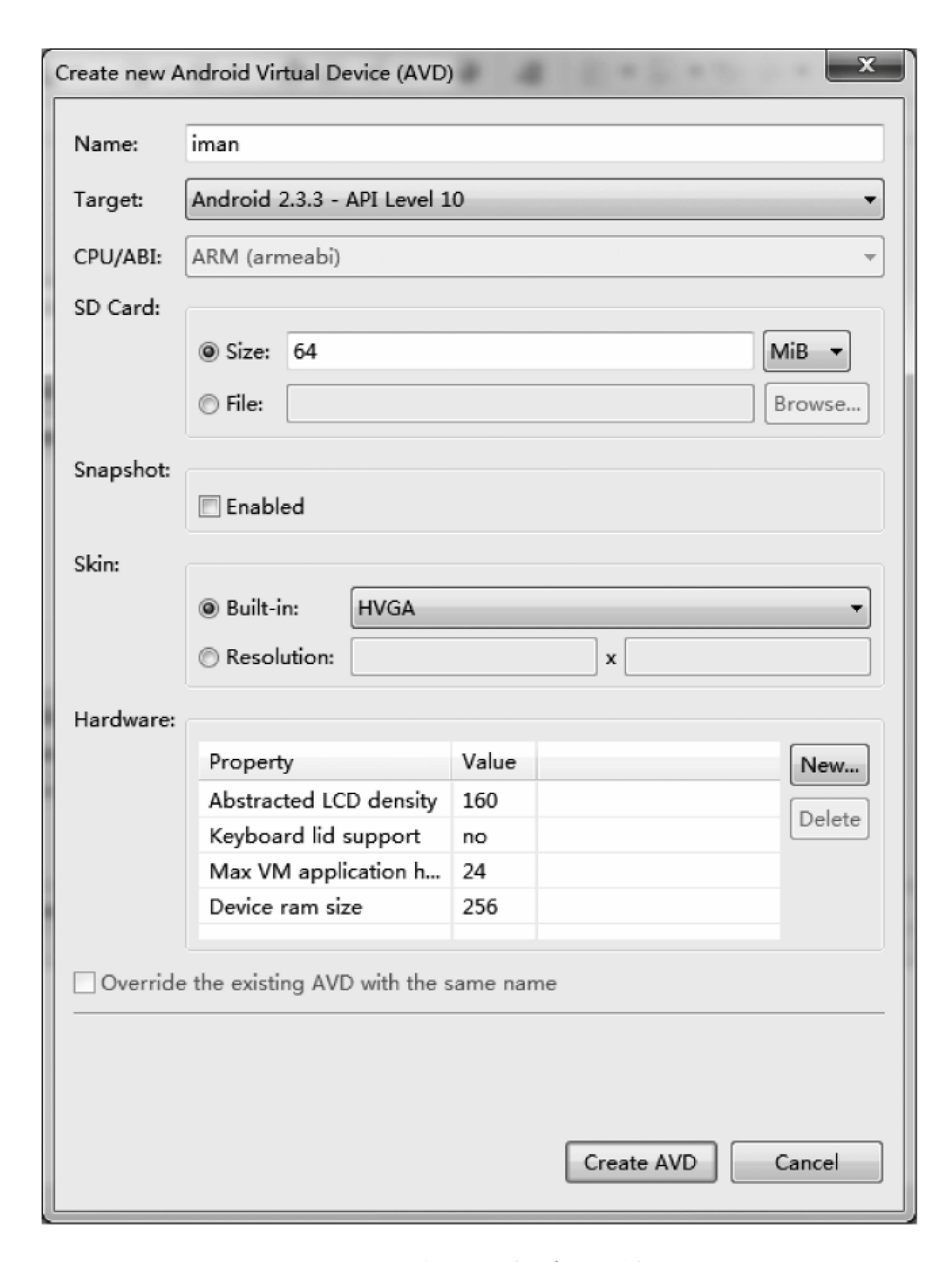


图 2.6 虚拟机创建对话框

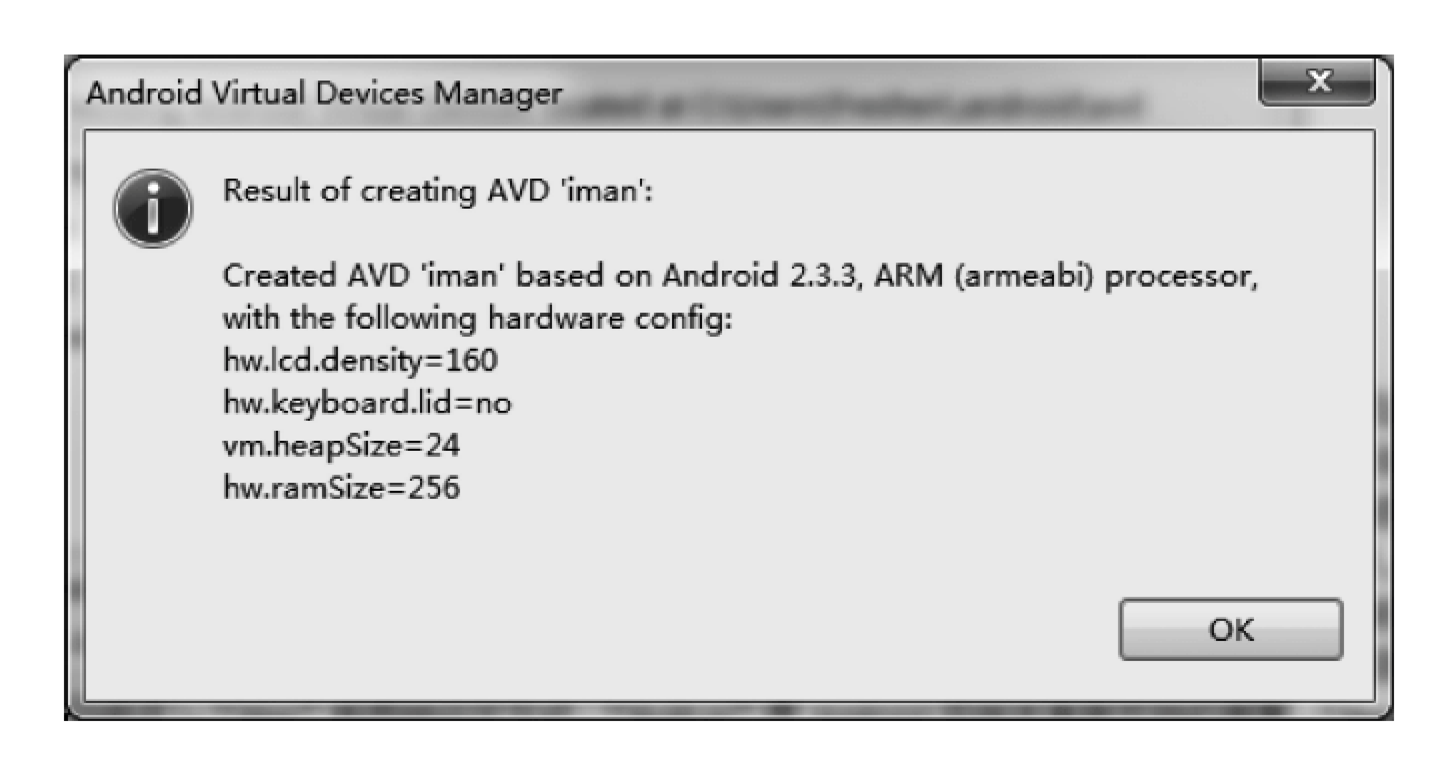


图 2.7 虚拟机创建成功

- 虚拟机创建时选定 Android 版本，可以测试与之匹配的应用程序。
- 虚拟机启动过程比较慢，一旦启动起来，测试阶段一般不需要再关闭。



图 2.8　Android 2.3.3 模拟器主界面

2.2.2　虚拟机的使用

Android 虚拟机的使用与真机类似，在手机界面上滑动鼠标就可以模拟滑屏操作。目前智能手机几乎已经普及大街小巷，对于简单操作就不再赘述。虚拟机中需要记住的操作有以下几个。

1. 修改系统语言

虚拟机默认的操作语言是英文，可以通过修改系统语言，更换成中文。在主界面上选择所有应用程序，打开 Settings(设置)，找到并打开 Language&Keyboard 子选项，打开 Select Language，向下一直滑动到列表底部，找到“中文(简体)”，就可以将系统语言更换为中文。更换后的系统设置界面如图 2.9 所示。

2. 通知栏的使用

在标题栏上向下拖动可以打开信息通知栏，如图 2.10 所示，通常情况下接收短信，完成网络下载任务，结束蓝牙传输等事件，会向任务栏发送一个通知，具体功能的实现后面会逐一介绍。

3. 虚拟机的按键

Android 虚拟器模拟准备了各种智能手机可能会出现的按键功能，主要有拍照、音量调

节、电源(并不会关掉虚拟机,只起到锁屏的作用)、通话按键、主界面按键(后面都称为Home 键)、菜单键(Menu)、退出键和搜索按键,中间是方向按键。




图 2.9 中文系统设置界面



图 2.10 系统任务通知栏

Android 虚拟机的功能非常强大,能够完成很多真机具备的功能。虚拟机可以直接联网(开发所用计算机需要联网),下载网上的 Android 应用程序,如图 2.11 所示。虚拟机的文字输入既可以借助手机界面的软键盘,也可以使用虚拟机的模拟键盘,还可以直接通过开发计算机的键盘直接输入。

虽然虚拟机的功能已经比较全面,基本可以完成常规应用程序的运行测试,但是虚拟机也有很多功能是无法实现的,比如不支持呼叫和来电,但是可以通过模拟实现,虚拟机无法实现耳机、USB、蓝牙等外连设备的扩展和连接,不能检测电池状态(标题栏上虽然有电池图标,但也只是一个图标而已,没有实际意义),不能进行 SD 卡的插拔(可以读写 SD 卡),无法 GPS 定位等,如果设计开发诸如此类的应用程序开发,需要借助真机进行测试。另外,虚拟机对于传感器类应用的开发也是力不从心。

为了模拟手机屏幕的横竖切换,虚拟机也可以实现屏幕的横竖变换,使用快捷键 Ctrl+F11 或 Ctrl+F12 即可。

关于虚拟机的创建与使用,需要注意以下几点。

- 虚拟机只能运行与之版本兼容的应用程序。如上面创建的虚拟机目标版本是 2.3.3,那么只有 Android 2.3.3 版本及以前的应用程序可开发运行。
- 虚拟机只能运行常规应用软件,无法模拟传感器以及真机具备的真实操作。不过这对于初学 Android 应用程序的开发已经足够了。
- 虚拟机启动比较慢,因此不要轻易关掉虚拟机。后面可能会遇到虚拟机无法响应的情况,只需要重置 adb 连接就可以。



图 2.11 虚拟机中的百度应用终端

2.2.3 真机测试

Android 提供的虚拟机可以解决一般应用程序的测试，但是与真机还是存在一定差距，比如在速度上与真机无法比拟，另外就是一些涉及真实操作的应用程序（如检测电池状态、GPS 定位等）虚拟机无法完成，因此真机测试是不可避免的，也是发布应用程序之前一定要完成的。

让真正的手机测试运行程序，需要在手机的设置面板中打开应用程序，找到开发项，勾选 USB 调试，使用 USB 数据线与开发计算机连接。如果是第一次连接，需要安装驱动程序，正确连接后在设备管理器窗口可以看到目前连接手机的信息，如图 2.12 所示。真机连接完成后，在 Eclipse 中开发的测试程序就可以直接在真机上运行了。



图 2.12 设备管理器中的真机连接

关于真机的使用需要注意以下几点。

- 真机只能运行与之操作系统兼容的应用程序。
- 使用真机测试，需要关闭虚拟机，否则会有 adb 连接错误。
- 真机可以测试的应用程序比较广泛，包括各种传感线、GPS 定位、电池状态等都可以进行。
- 刚开始接触 Android 开发，如果没有 Android 手机，不用以此为借口，购买新手机，至于是否需要购买，完全可以等入门后再决定。

2.3　解析 Android 项目结构

到目前为止，已经安装并配置了 Android 应用程序的开发环境，创建 Android 虚拟机或使用真机进行应用程序的测试。本节将创建第一个 Android 应用程序，并用于测试以上开发环境和虚拟机是否能正常工作。

2.3.1　创建 Android 项目

启动 Eclipse，配置工作空间（项目所在文件夹），配套资料中 BookDemo 文件夹是本书所列举项目的工作空间，完整的源代码可以在配套资料中查找到。如果是第一次启动 Eclipse，会出现欢迎界面，如果想了解 Eclipse 的相关知识，可以通过欢迎界面进行学习，否则可以直接关闭（以后将不再显示）。

依次选择 File→New→Other 命令，打开"新建项目"窗口，如图 2.13 所示。找到并展开 Android 项，选择 Android Project（如果升级到 ADT4.2 会改为 Android Application Project），单击 Next 按钮，打开"新建 Android 项目"窗口，输入项目的名称，比如 Part02，如图 2.14 所示。项目名称下面的单选项说明如下。

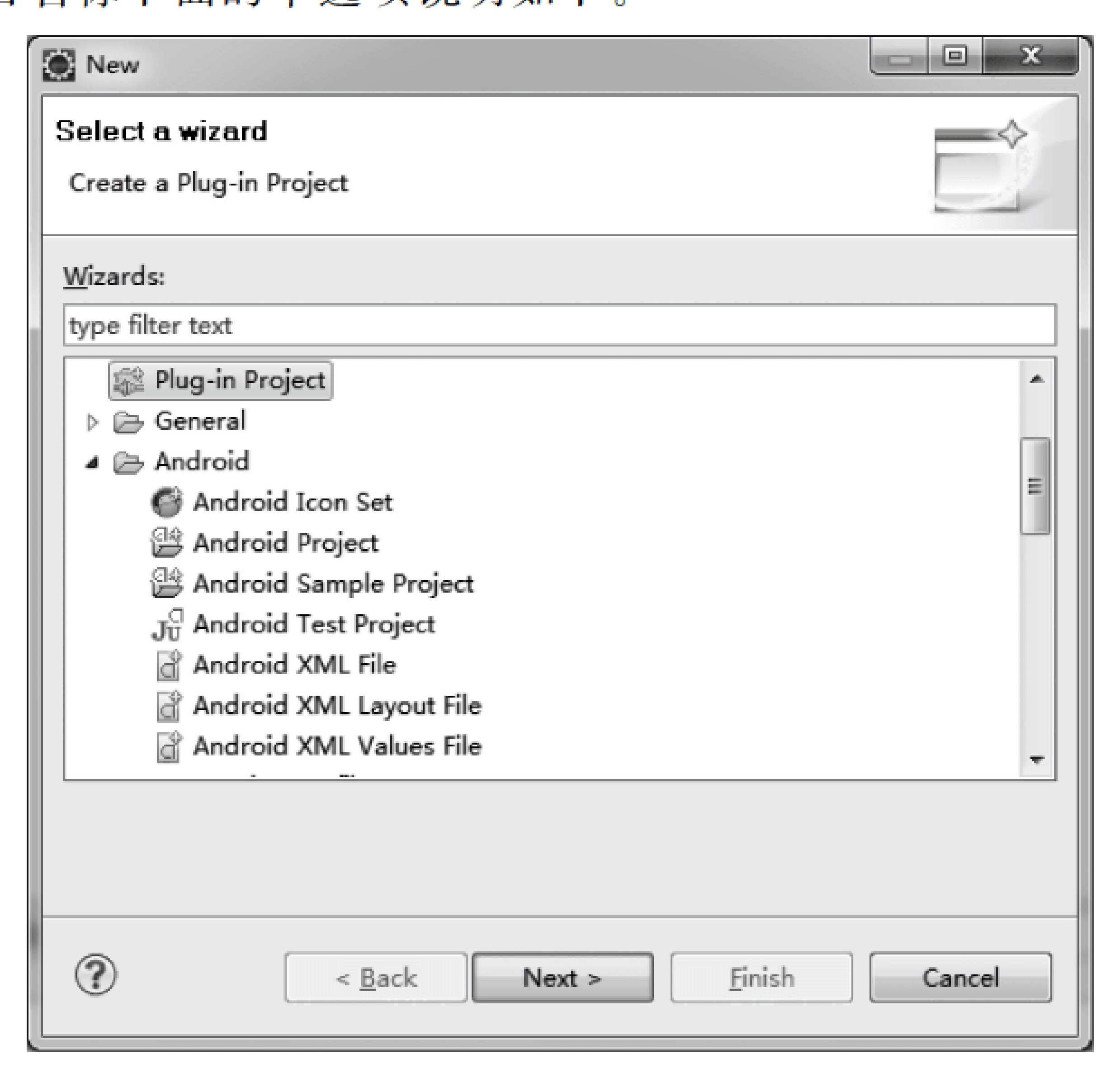


图 2.13　"新建项目"窗口

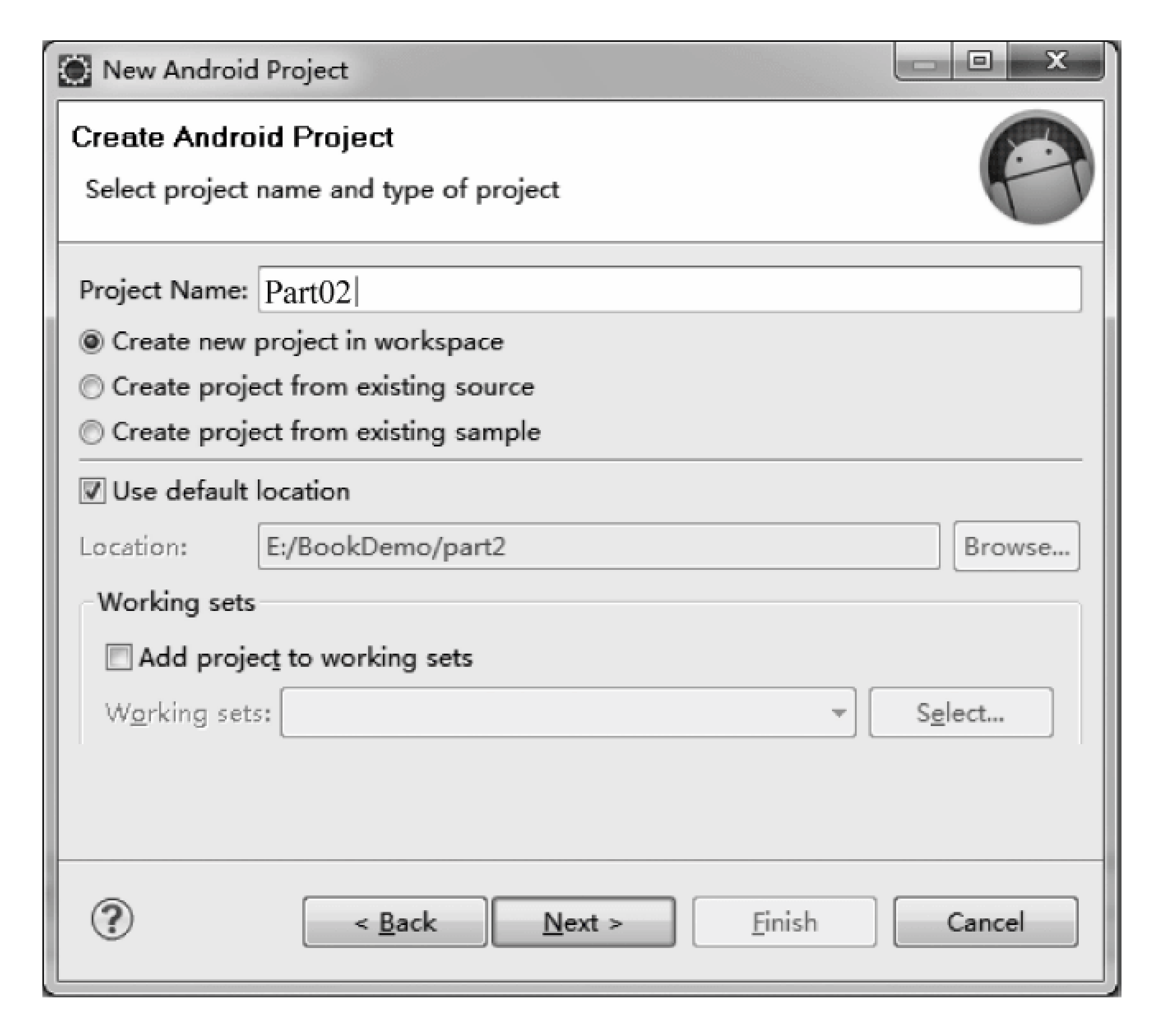


图 2.14 “新建 Android 项目”窗口(一)

(1) Create new project in workspace:在工作空间中创建新项目,这是默认项。

(2) Create project from existing source:通过已存在的资源创建项目。

(3) Create project from existing sample:通过例子创建项目。

Use default location,是选定当前工作空间,这不需要修改。Working sets,是否将项目添加到工作集中,这个选项不需要修改,保持默认值即可。

完成设置后,单击 Next 按钮,选择项目的工作版本,如图 2.15 所示。此处选择 Android 2.3.3,与 2.2 节中创建虚拟机的工作版本一致,这是 Android Open Source Project,是开源项目,对应 API 版本号是 10,这个版本号在项目中也会出现,决定了可以运行在操作系统的哪个版本上。确定操作系统的版本后,单击 Next 按钮,打开最后一个配置窗口,如图 2.16 所示。

Application Info 即应用程序信息,此处配置的内容与应用程序的内容有关。Application Name 是应用程序的名称,将来安装到手机上可以显示。Package Name(包名)必须是两段以上(比如 edu.freshen.code 是三段),否则会提示 Package name must have at least two identifiers。Create Activity 默认是勾选,由应用程自动创建一个 Activity。Minimum SDK 为最低版本号。这里所设置的信息将应用到程序内,需要谨慎设置,完成后单击 Finish 按钮,结束创建新 Android 项目的全部过程。

Android 应用项目的创建过程比较简单,需要遵循以下几条建议。

- 不同 ADT(Eclipse 的插件),新建 Android 项目的过程不同,但需要注意设置的参数以上都有说明。
- 对于版本信息应该事先确定,一般选择 2.1 或 2.3 即可,这样可以使得更多的手机兼容应用程序。
- 设置包名时,最少需要两段,可以根据自己的开发习惯命名。

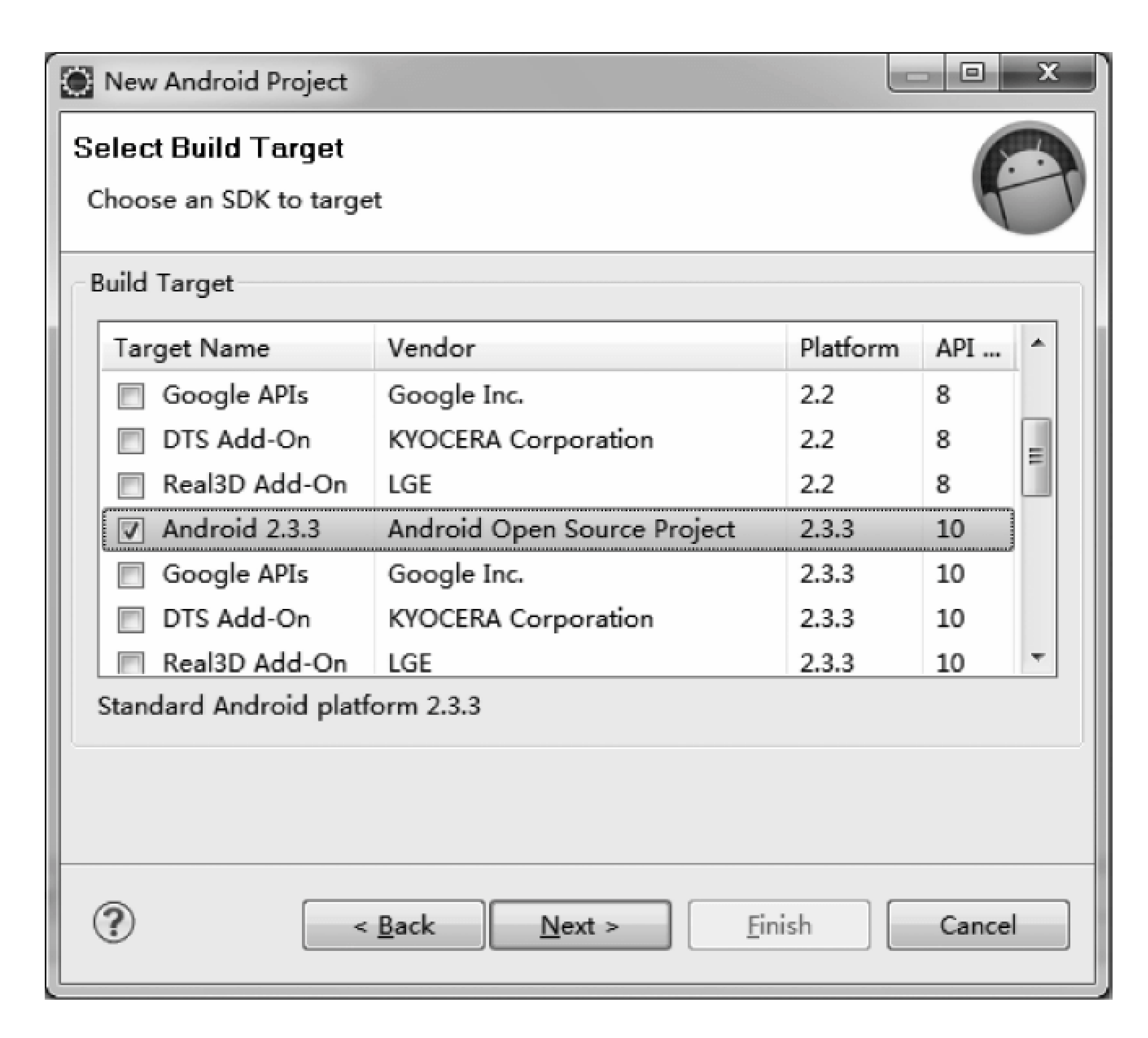


图 2.15 新建 Android 项目窗口(二)

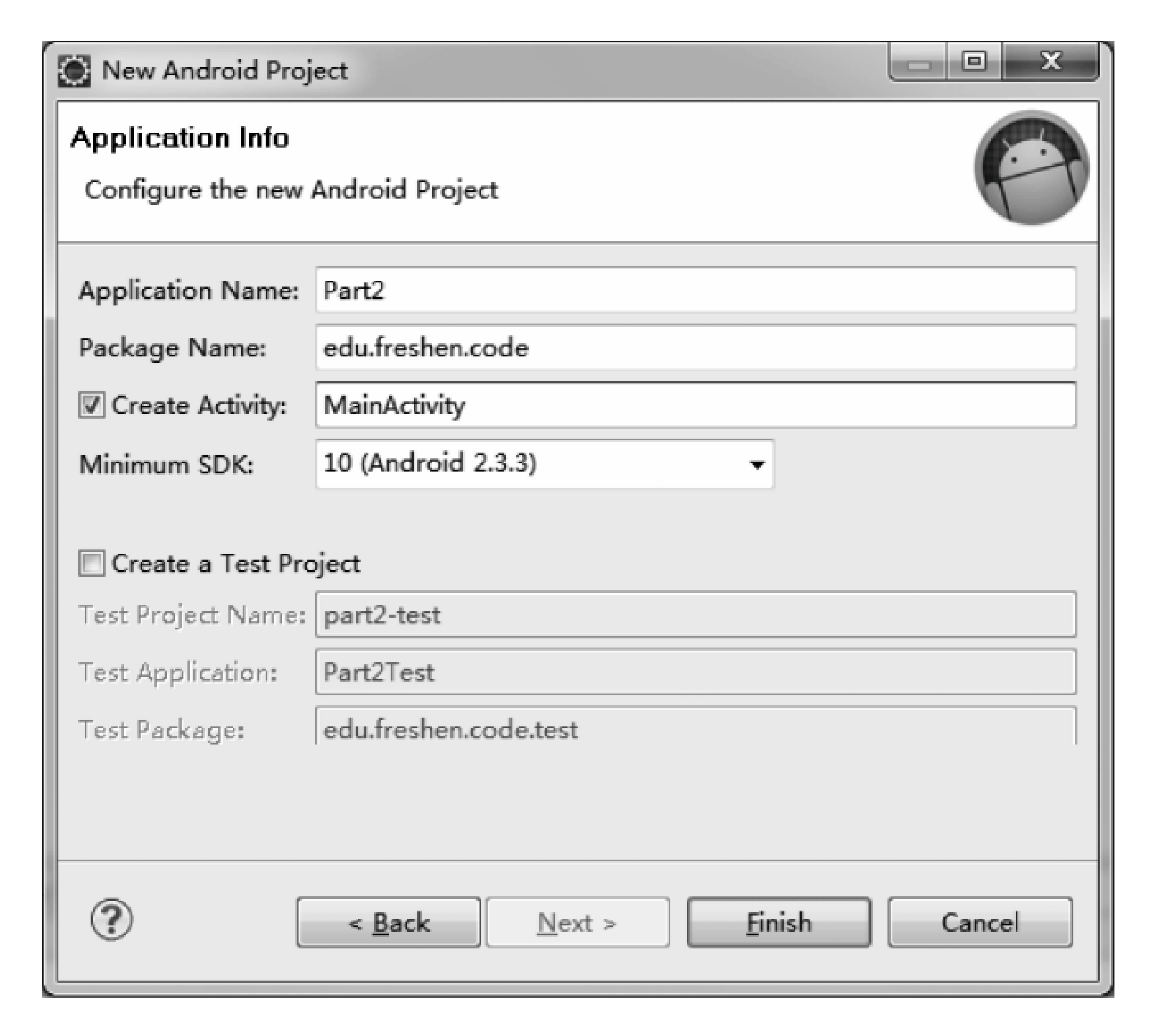


图 2.16 新建 Android 项目窗口(三)

2.3.2 Android 项目结构介绍

Android 项目创建后,可以在 Package Explorer(包视图)中看到整个项目的所有结构,如图 2.17 所示。这个项目的结构比 Java 应用程序的项目结构要复杂得多,这是由于

Android 应用程序的编译运行机制不同。以下对该项目的主要结构进行介绍。

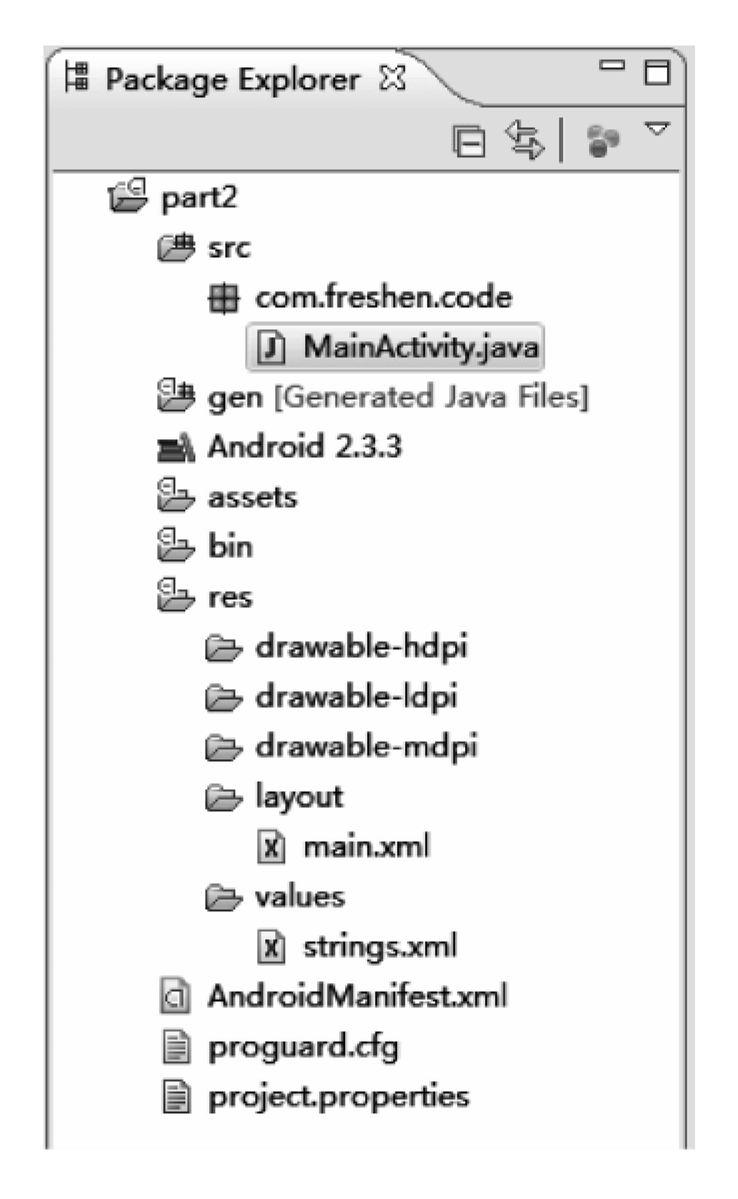


图 2.17　Android 项目结构

1. src

源代码资源，与 Java 项目的 src 类似，存放 Android 应用程序的代码。其中的包(com.freshen.code)和类(MainActivity.java)是在项目创建过程中已经设定，Eclipse 自动生成的。

2. gen

Generated Java Files 表明这是一个由系统自动生成的文件。展开后有 R.java 类，该类是自动生成，禁止修改，用于引用后面 res 中的资源，当 res 中文件发送变化后，R.java 会自动修改相应的值。

3. Android 2.3.3

系统类库，由创建项目时选定的版本决定。

4. assets

存放外部资源，可以与应用程序打包在一起，与后面 res(也是用于存放资源)不同的是，res 中的资源可以与 gen 中的 R 类对应，assets 中的资源不会在 R 类中有引用。

5. bin

存放编译后类文件。

6. res

Android 应用程序的资源包，默认有 5 个子文件夹，drawable-××××用于存放图片资源，为了兼容不同机型屏幕的分辨率，图片资源文件夹又划分为以下三个。

（1）drawable-hdpi：高分辨率图片，如 WVGA 屏幕（480×800），FWVGA（480×854）。

（2）drawable-ldpi：低分辨率图片，如 QVGA 屏幕（240×320）。

（3）drawable-mdpi：中分辨率图片，如 HVGA 屏幕（320×480）。

layout 文件夹用于存放屏幕布局文件，其中 main. xml 文件是 MainActivity 默认采用的布局。values 文件夹可以存放多种类型资源，默认只有 strings. xml 文件，存放字符串，除此之外还有：

（1）array. xml 可以存放数组。

（2）colors. xml 可以存放颜色的字符串值。

（3）dimens. xml 可以存放尺寸值（dimension value）。

（4）styles. xml 存放样式（style）对象。

res 中还可以出现的文件有：

（1）anim 文件夹用于存放动画文件，可以被编译进逐帧动画（frame by frame animation）或补间动画（tweened animation）对象。

（2）xml 文件夹存放任意的 XML 文件。

（3）raw 文件夹存放直接复制到设备中的任意文件。

Android 应用程序布局的方式类似 Web 开发中采用 CSS 样式，将布局文件和值的引用与源代码分离，res 文件夹就是用于存放布局和引用值的文件夹，放入其中的所谓资源，都会生成一个 ID，对应到 gen 文件夹的 R 类中。在程序开发过程中，可以直接使用 R 类中的静态属性，引用 res 中的任意资源。代码 02-1 显示的 R. java 中代码，它们与 res 中的资源具有一一对应的关系。

📖代码 02-1

```
package com.freshen.code;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher = 0x7f020000;
    }
    public static final class layout {
        public static final int main = 0x7f030000;
    }
    public static final class string {
        public static final int app_name = 0x7f040001;
        public static final int hello = 0x7f040000;
    }
}
```

R 中的内部类都是静态的，可以通过 R. 直接引用。drawable 类对应 res 中 drawable 文

件夹，属性 ic_launcher 对应 res 中 drawable-××××（不区分分辨率）中的图片资源 ic_launcher.png。layout 类对应 res 中 layout 文件夹，属性 main 对应 main.xml 布局文件。string 类对应 res 中 valuse 文件夹中 strings 文件，属性 app_name 和 hello 分别是 strings.xml 中的两个字符串。

双击打开 strings.xml 文件，如图 2.18 所示，默认打开 Resources 视图，在左侧 Resources Elements 列表中有 R.java 中 string 内部类的两个属性。切换视图为strings.xml，显示代码格式，见代码 02-2。

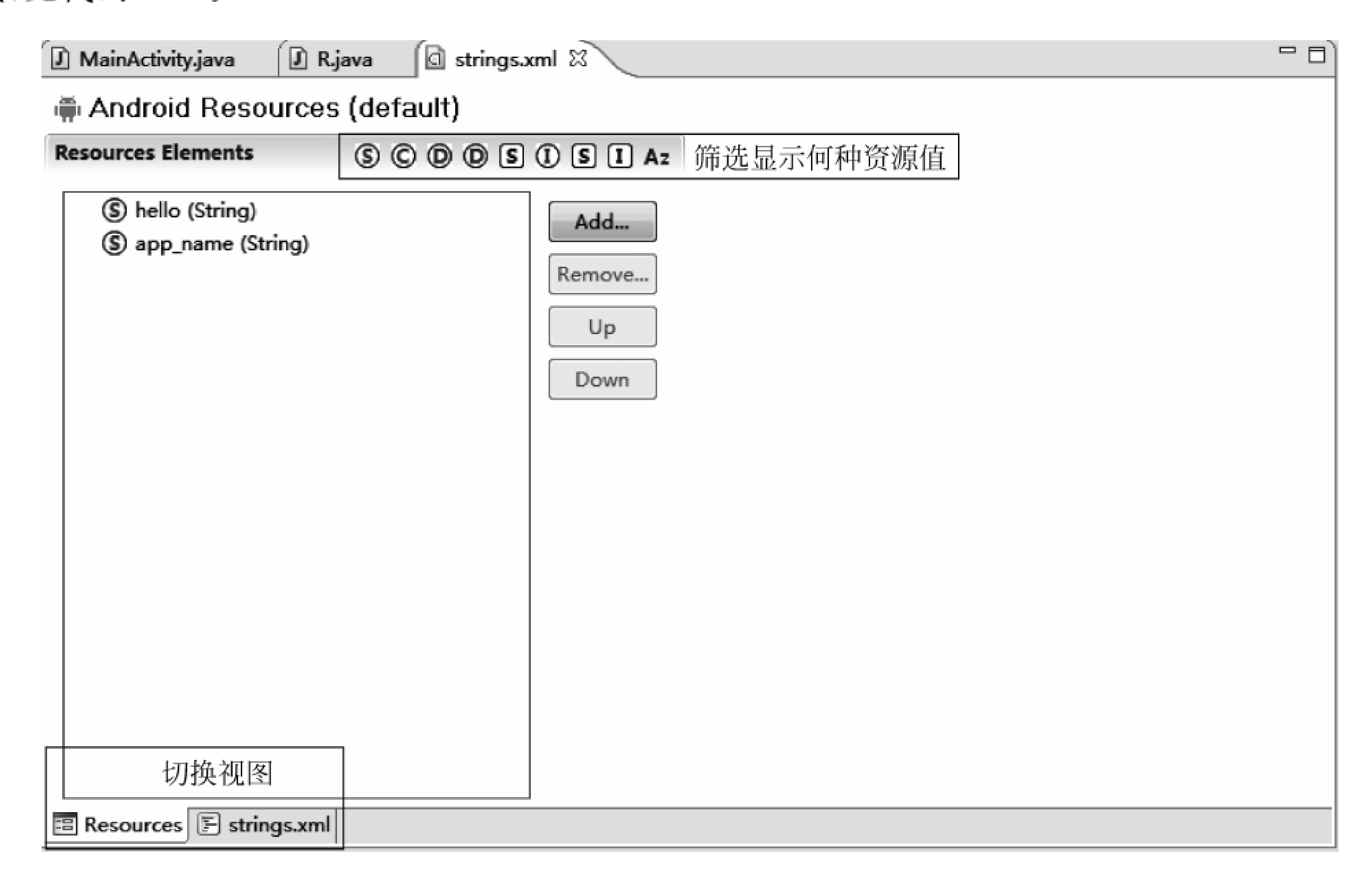


图 2.18 strings.xml 资源文件

代码 02-2

```
<?xml version = "1.0" encoding = "utf - 8"?>
<resources>

    <string name = "hello"> Hello World, MainActivity!</string>
    <string name = "app_name"> Part02 </string>          字符串的引用和值
</resources>
```

7. AndroidManifest.xml

AndroidManifest.xml 是每个 Android 应用程序中必需的文件。可以设置应用程序的编码格式，设置 ICON 图标等，能够声明应用程序中的 Activities（活动）、ContentProviders（内容提供）、Services（服务）和 Intent Receivers（意图/广播接收），还可以指定 permissions（权限）。双击打开 AndroidManifest.xml 文件，如图 2.19 所示。通过底部切换视图模式到 AndroidManifest.xml 视图，如代码 02-3 所示。

图 2.19 AndroidManifest.xml 视图

代码 02-3

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.freshen.code" android:versionCode="1" android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="10" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

manifest 标签中定义项目的属性，如包名和版本。uses-sdk 标签定义最低 SDK 版本。application 标签定义应用程序的属性和功能，如应用程序的图标、名称。activity 标签是活动界面，本项目中只有一个可视界面(activity)，所以 AndroidManifest.xml 中只有一个 activity 标签的声明，如果设计了多个可视活动界面(activity)，这里需要配置多个 activity (就这一点而言，很像配置 JSP 开发中的 Servlet)。activity 标签中有 name 属性指定这个活动视图由哪个类实现(注意前面有个.，与包名连在一起，可以确定所使用类)，label 属性指

明这个活动视图的标题(类似于窗口的 Title)。intent-filter 为意图过滤器，此处的作用是确定所在 activity 为主 activity(会第一个启动)。

8. proguard.cfg 与 project.properties

proguard.cfg 用于防止发布的 apk 文件被反编译，Android 2.3 版本以后有此功能。project.properties 是项目参数，无须改动。

初识 Android 开发，无须过分关注细节，Android 项目的结构比较多而且复杂，需要注意以下几条建议。

- src 中用于组织应用程序的代码。
- gen 文件夹中 R 类用于引用资源，禁止编辑。
- res 中存放各种资源，在 R 中生成相应的引用，便于代码中使用。
- AndroidManifest.xml 文件非常重要，需要配置的内容也多，但现在可以不关注。

2.3.3 运行结果分析

Android 项目创建完成后，自动生成 MainActivity.java 类，类名是在创建过程中指定的，打开该类，如代码 02-4 所示。MainActivity 继承 Activity 类，并重写 onCreate 方法，该方法会在 Activity 初次创建时调用，常用于 Activity 界面的布局初始化、组件初始化。

代码 02-4

```
package com.freshen.code;
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
                        //将 Activity 视图/布局设置为 R.layout.main 所对应的布局文件
    }
}
```

在项目 Part02 上单击右键，选择 Run as→Android Application 命令。如果虚拟机没有启动，此时 Eclipse 会启动虚拟机，如果虚拟机已经启动，Eclipse 会将项目打包成 apk 文件，安装到虚拟机，并运行，如图 2.20 所示，标题为 Part02，内容是一行字符串。在虚拟机应用程序窗口查看项目，显示如图 2.21 所示。下面解析项目运行后的几个问题。

1. 界面显示字符串的由来

由配置文件 AndroidManifest.xml 中 intent-filter 标签指定的 Activity 为主界面，即 MainActivity 所对应的类将最先启动，onCreate 方法被执行，其中影响显示布局的 setContentView 方法，将 Activity 布局设置为 R.layout.main 所对应的布局文件。找到 res 文件夹 layout 文件夹中的 main.xml 文件，双击打开，如图 2.22 所示。



图 2.20 Part02 项目运行效果



图 2.21 Part02 应用程序图标

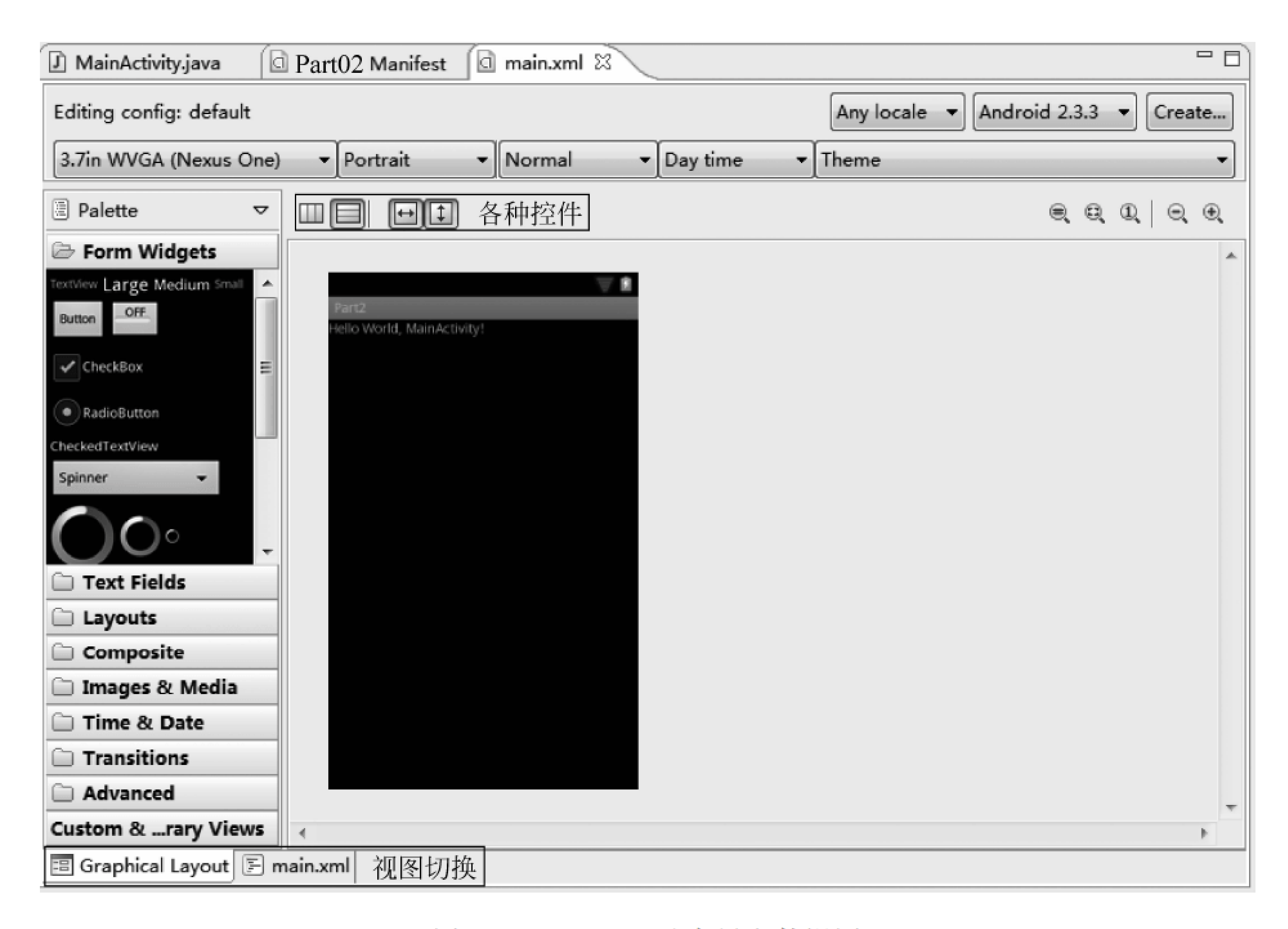


图 2.22 main.xml 布局文件视图

Android 布局视图中左边提供了丰富的 Android 应用控件，可以直接拖动到右侧的 Activity 视图中，这些控件在后续的学习中会有所介绍。上边是针对不同设备所具有布局控件的设定。右侧显示当前 Activity 布局情况，可以通过右上侧的放大按钮放大。下边可以切换视图为代码模式，如代码 02-5 所示。

📖 **代码 02-5**

```
<?xml version = "1.0" encoding = "utf - 8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "fill_parent"
    android:layout_height = "fill_parent"
    android:orientation = "vertical" >
    <TextView
        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:text = "@string/hello" />
</LinearLayout>
```

运行效果由这个布局文件决定，这是项目创建后自动生成的一个布局文件，可以修改，也可以另行创建其他布局（后面会有介绍）。LinearLayout 标签是线性布局管理器（Android 中有很多布局管理器，与 Java 中开发 GUI 应用时布局管理器不同），指明放入这个标签中的控件将采用线性布局，属性 layout_width 标明布局的宽度，值 fill_parent 标明布局将会充满父容器（Activity 的视图大小）；属性 layout_height 标明布局高度；属性 orientation 标明线性布局的方向，值 vertical 标明方向为竖直方向。

布局管理器中只有一个控件 TextView（文本控件），宽度为填充父容器（TextView 的父容器是 LinearLayout），高度是与 TextView 中出现的内容匹配，属性 text 是该控件要显示的文本信息，值@string/hello 标明要引用资源文件 strings.xml 中名为 hello 的字符串（参考代码 02-2）。

2. 标题

查看 AndroidManifest.xml 文件（参考代码 02-3），可以找到下面的代码。属性 label 指明该 Activity 的标题值是什么，值@string/app_name 是引用名为 app_name 的字符串。

```
android:label = "@string/app_name" >
```

3. 应用程序的图标与名字

查看 AndroidManifest.xml 文件（参考代码 02-3），可以找到下面的代码。属性 icon 指明该应用程序的图标，值@drawable/ic_launcher 为引用 res 中 drawable-××××文件夹中的图片资源。注意，属性值会影响所在标签的特征，比如同样的属性 label，位于 activity 标签中，标明 activity 的标题，位于 application 标签，则标明该应用程序的名称。

```
android:icon = "@drawable/ic_launcher"
android:label = "@string/app_name" >
```

2.4 应用程序与 Activity

Activity 是 Android 应用程序的基本组成之一，主要用于显示应用程序的界面，但它不同于 J2SE 中 JFrame（或 Frame）的概念。如果把手机屏幕比作是一个浏览器，Activity 可以被视为其中一个网页，呈现各种视图、组件。

2.4.1　Activity 介绍

在应用程序创建的过程中，选择 Create Activity 复选框（见图 2.16），Eclipse 会自动创建一个 Activity，并把它作为主 Activity（首先启动界面）。Android 应用程序可以由多个 Activity 组成，就如同一个 Web 应用程序由多个页面组成一样，Activity 之间可以相互跳转，并传递参数。在某一时刻手机屏幕只能显示一个 Activity，其他 Activity 处于不可见状态。某个 Activity 启动，则之前的 Activity 就要被挤到下面。比如，在操作游戏时，呈现游戏界面的 Activity 处于最上层（用户可见），在有电话接入时，通话界面 Activity 将会启动，并处于最上层，此时游戏界面 Activity 会被挤到下一层，通话结束后，通话界面 Activity 被销毁，处于下一层的游戏界面 Activity 会提升到最上层。

Android 系统中有很多应用程序，每个应用程序可以有多个 Activity，它们的维护工作由 Android 系统完成，使用 Activity 栈管理已经启动并活着的 Activity（当 Activity 销毁后会从栈中撤出）。新启动的 Activity 被存放于栈中，位于栈顶，获得输入焦点，用户可见。在当前活动的 Activity 上按返回（Back）键，该 Activity 从栈中取出，然后销毁，然后下一个 Activity 移动到栈顶，被恢复为可见状态。

对于一个游戏应用程序，一般会包括如下三个 Activity，它们在 Activity 栈中的状态如图 2.23 所示。

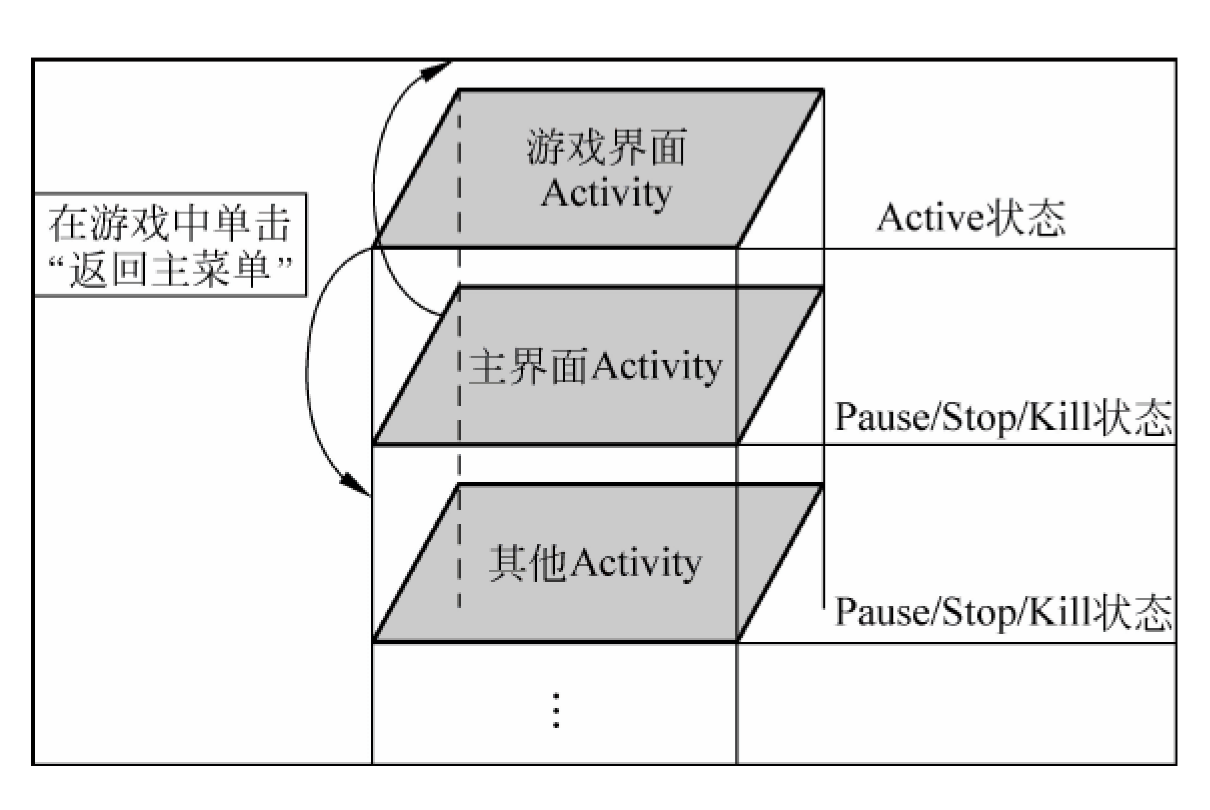


图 2.23　Activity 栈示意图

（1）主界面 Activity，在该界面上用户可以选择开始新游戏、继续、查看帮助或设置等功能。

（2）游戏界面 Activity，游戏应用程序的核心部分，用户操作游戏。

（3）设置界面 Activity，用户可以完成设置难度、关闭音乐等操作。

除了最顶层 Activity 外，其他的 Activity 都有可能在系统内存不足时被回收（Kill 或 Destroy），Activity 越是处在栈的底层，被系统回收的可能性越大。Android 系统负责管理栈中的 Activity 对象，根据 Activity 所处的状态来改变其在栈中的位置。Activity 从创建、显示、隐藏到销毁都有各自的生命周期，当状态发生改变时，处于生命周期中的方法将被执行。

2.4.2　Activity 的生命周期

了解 Activity 的生命周期对于开发 Android 应用程序大有裨益，掌握生命周期中各状态转

换时调用的方法对开发者来说至关重要。涉及生命周期中状态改变时的方法有以下7个。

(1) onCreate：Activity第一次被创建时调用，可以在该方法中设定布局，初始化控件，绑定数据，设置监听器。

(2) onRestart：当Activity停止后又重新显示被调用，如果Activity是通过onCreate第一次创建来的，该方法不调用。无论哪种情况，onRestart执行结束都会调用onStart方法。

(3) onStart：Activity将要被用户可见(还没有可见)时调用，可以开启线程操作。

(4) onResume：用户可以与Activity交互时调用，此时Activity处于Activity栈顶部。可以在该方法中启动音频、视频和动画。

(5) onPause：这是比较重要的一个方法，当其他Activity启动后，还没有显示出来时调用，可以在该方法中停止线程、动画等占有CPU资源的任务，提交没有保存的数据。

(6) onStop：当其他Activity启动，并可见后，如果能遮住本Activity，则会调用此方法，如果没有遮住本Activity(比如其他Activity是Dialog模式)，则不会调用此方法。该方法执行后，如果Android系统检测内存不足，本Activity会被Kill。

(7) onDestroy：当Activity被销毁前调用的最后一个方法。

当一个Activity对象被创建、销毁或者启动另一个Activity时，它会在Active、Pause、Stop和Kill 4种状态之间进行转换，这种转换的发生依赖于用户的动作。图2.24说明了Activity在不同状态间转换的时机和条件。

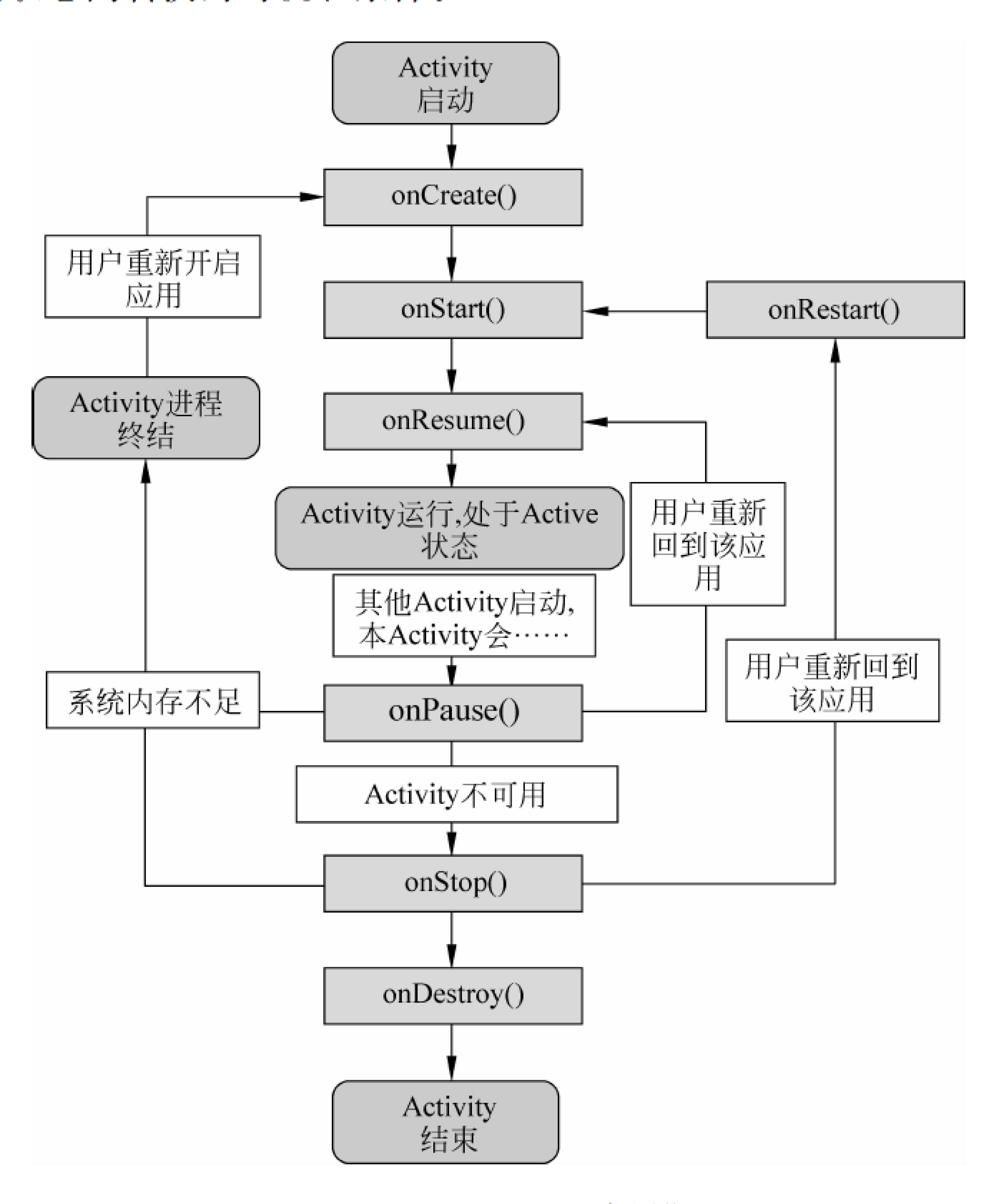


图2.24 Activity生命周期

代码 02-6 是对上面新建项目的修改,添加了 Activity 的生命周期方法,使用 Log.i(Android 程序的调试方法,后面有详细介绍,如果此处想测试代码,可以先阅读 2.5 节的内容)向日志中添加提示信息,完整的代码请参考配套资料 Part02 项目。

代码 02-6

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.i("MainActivity", "onCreate");
}
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.i("MainActivity", "onDestroy");
}
@Override
protected void onPause() {
    super.onPause();
    Log.i("MainActivity", "onPause");
}
@Override
protected void onRestart() {
    super.onRestart();
    Log.i("MainActivity", "onRestart");
}
@Override
protected void onResume() {
    super.onResume();
    Log.i("MainActivity", "onResume");
}
@Override
protected void onStart() {
    super.onStart();
    Log.i("MainActivity", "onStart");
}
@Override
protected void onStop() {
    super.onStop();
    Log.i("MainActivity", "onStop");
}
```

运行该项目,并按虚拟机上的 Back(退出)键,查看 Log 视图的输出信息,如图 2.25 所示。重新运行该项目,按虚拟机上 Home(主界面)键,查看 Log 视图信息,显示如图 2.26 所示。

两次输出结果对比可以得出这样的结论:两次重新运行都是遵循 onCreate、onStart、onResume 过程,都是第一次创建该 Activity,所以 onRestart 并没有执行。按 Back 键可以直接退出该 Activity,这由 onDestroy 方法的执行可以得出。按 Home 键,onDestroy 方法

不执行，该 Activity 被手机主界面 Activity 挤入 Activity 栈。在虚拟机上找到该应用程序图标(不用在 Eclipse 中重新执行 run as 命令)，运行该应用程序，查看 LogCat 输出，如图 2.27 所示。在 onStop 方法执行后，执行 onRestart 方法，应用程序 Activity 重新回到可见，由此可见，Home 键并不会退出应用。

L...	Time	PID	Application	Tag	Text
I	03-23 06:58:13.300	373		MainActivity	onCreate
I	03-23 06:58:13.300	373		MainActivity	onStart
I	03-23 06:58:13.330	373		MainActivity	onResume
I	03-23 06:58:25.701	373	com.freshen.code	MainActivity	onPause
I	03-23 06:58:26.280	373	com.freshen.code	MainActivity	onStop
I	03-23 06:58:26.280	373	com.freshen.code	MainActivity	onDestroy

图 2.25 LogCat 视图输出信息(一)

L...	Time	PID	Application	Tag	Text
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onCreate
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onStart
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onResume
I	03-23 07:05:08.119	373	com.freshen.code	MainActivity	onPause
I	03-23 07:05:08.690	373	com.freshen.code	MainActivity	onStop

图 2.26 LogCat 视图输出信息(二)

L...	Time	PID	Application	Tag	Text
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onCreate
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onStart
I	03-23 07:05:03.410	373	com.freshen.code	MainActivity	onResume
I	03-23 07:05:08.119	373	com.freshen.code	MainActivity	onPause
I	03-23 07:05:08.690	373	com.freshen.code	MainActivity	onStop
I	03-23 07:08:15.390	373	com.freshen.code	MainActivity	onRestart
I	03-23 07:08:15.390	373	com.freshen.code	MainActivity	onStart
I	03-23 07:08:15.390	373	com.freshen.code	MainActivity	onResume

图 2.27 LogCat 视图输出信息(三)

修改 Part02 项目，实现从 MainActivity 界面跳转到其他界面的功能，修改 MainActivity.java 文件中的代码如代码 02-7 所示，修改 main.xml 布局文件，添加两个按钮控件，如代码 02-8 所示。添加 FullScreenActivity.java 和 DialogActivity.java，继承 Activity 类，作为另外两个界面，并新建与之对应的布局文件 full.xml 和 dialog.xml。该项目的完整代码参考配套资料 Part02 项目。

代码 02-7

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.i("MainActivity", "onCreate");
    //获取控件并注册监听器
```

```
        Button bt1 = (Button) findViewById(R.id.button1);
        Button bt2 = (Button) findViewById(R.id.button2);
        bt1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                Intent intent  = new Intent(MainActivity.this,FullScreenActivity.class);
                MainActivity.this.startActivity(intent);
            }
        });
        bt2.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                Intent intent  = new Intent(MainActivity.this,DialogActivity.class);
                MainActivity.this.startActivity(intent);
            }
        });
    }
```

代码 02-8

```
    <Button
            android:id="@+id/button1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="全屏 Activity" />
    <Button
            android:id="@+id/button2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Dialog Activity" />
```

由于添加了两个新的 Activity，需要在 AndroidManifest.xml 文件中增加配置信息如代码 02-9 所示，新添加的代码位于原来 Activity 标签的后面即可，其中属性 theme 指明该 Activity 是以对话框的形式呈现。

代码 02-9

```
    <activity
        android:name=".FullScreenActivity"
    />
    <activity
        android:name=".DialogActivity"
        android:theme="@android:style/Theme.Dialog"
    />
```

再次运行项目，效果如图 2.28 所示，单击按钮可以跳转到其他 Activity。FullScreenActivity 是可以遮住 MainActivity 的全屏界面，DialogActivity 是无法遮住 MainActivity 的对话框界面，如图 2.29 所示。

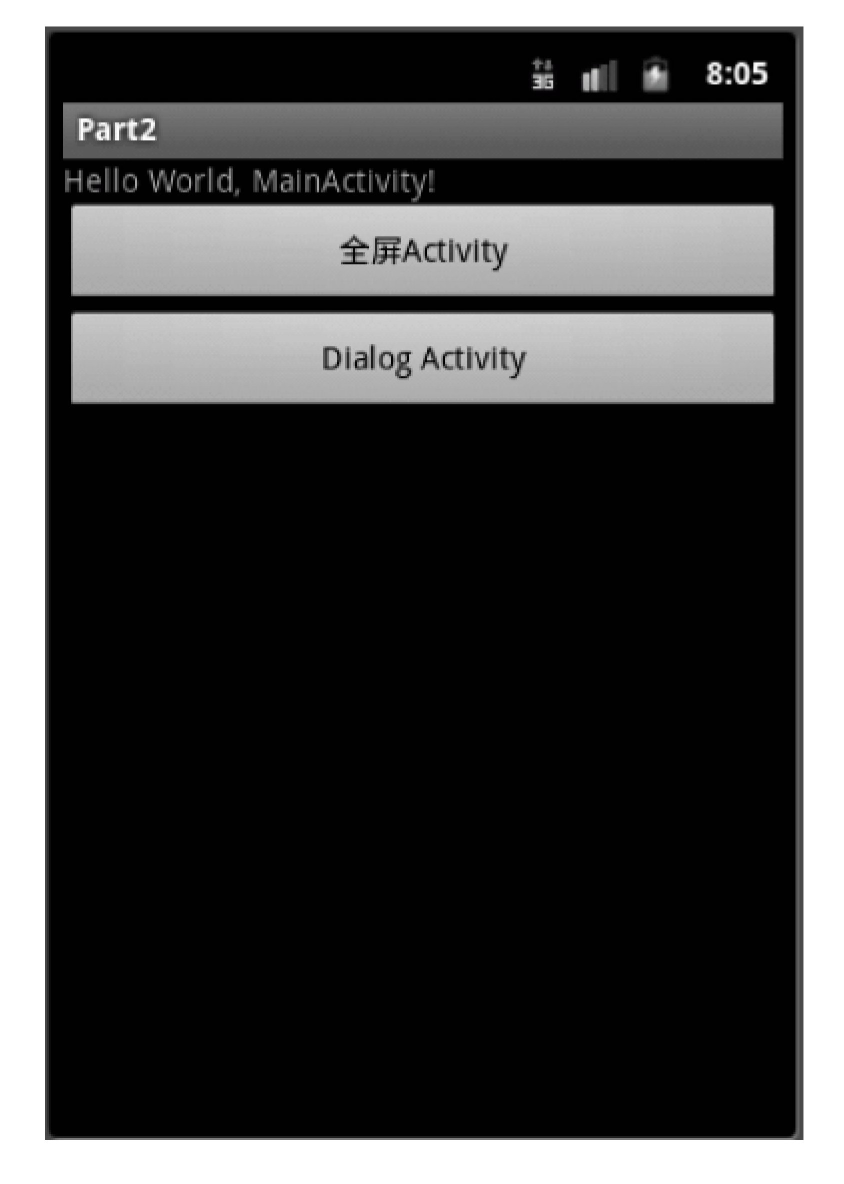



图 2.28　Part02 项目运行界面



图 2.29　对话框模式 Activity

目前，Part02 应用程序具有三个 Activity，为了测试 MainActivity 的生命周期方法，分别完成从 MainActivity 跳转到其他 Activity，并按 Back 键返回到 MainActivity，输出信息如图 2.30 与图 2.31 所示。

L...	Time	PID	Application	Tag	Text
I	03-23 08:15:44.730	492	com.freshen.code	MainActivity	onCreate
I	03-23 08:15:44.730	492	com.freshen.code	MainActivity	onStart
I	03-23 08:15:44.741	492	com.freshen.code	MainActivity	onResume
I	03-23 08:15:53.360	492	com.freshen.code	MainActivity	onPause
I	03-23 08:15:53.820	492	com.freshen.code	MainActivity	onStop
I	03-23 08:15:56.180	492	com.freshen.code	MainActivity	onRestart
I	03-23 08:15:56.180	492	com.freshen.code	MainActivity	onStart
I	03-23 08:15:56.180	492	com.freshen.code	MainActivity	onResume

图 2.30　遮住 Activity 返回的输出信息

L...	Time	PID	Application	Tag	Text
I	03-23 08:17:22.120	492	com.freshen.code	MainActivity	onCreate
I	03-23 08:17:22.180	492	com.freshen.code	MainActivity	onStart
I	03-23 08:17:22.180	492	com.freshen.code	MainActivity	onResume
I	03-23 08:17:24.200	492	com.freshen.code	MainActivity	onPause
I	03-23 08:17:27.060	492	com.freshen.code	MainActivity	onResume

图 2.31　未遮住 Activity 返回的输出信息

从 MainActivity 跳转到 FullScreenActivity 时，MainActivity 执行 onPause 和 onStop 方法，被挤入栈中，FullScreenActivity 呈现。当 FullScreenActivity 执行 Back 操作后，会被销毁，从栈中撤出，MainActivity 执行 onRestart 再次呈现。

从 MainActivity 跳转到 DialogActivity 时，MainActivity 执行 onPause，但没有执行 onStop，此时用户可以同时看见 DialogActivity 和 MainActivity，如图 2.29 所示。当在

DialogActivity 执行 Back 操作后，界面被销毁，从栈中撤出，MainActivity 执行 onResume 方法获得焦点并呈现。

以上分别介绍了按手机键和应用程序内部界面切换，对 Activity 生命周期状态的影响。对应前面的介绍需要注意以下几条建议。

- onCreate 方法只有在 Activity 第一次创建时执行，可以用于初始化操作。
- onResume 方法执行后，用户就可以操作界面，在此可以启动线程，播放声音、动画等。
- onStop 方法执行后，程序有可能会被结束，在此需要回收系统资源，提交数据。
- Android 中很多类如果翻译成汉语会比较奇怪，因此对于 Android 中的重要组件不作翻译。
- Eclipse 中重写父类方法的快捷键是 Shift＋Alt＋S 键。
- 以上功能的完整代码在配套资料 Part02 中。

2.4.3 Intent 与 Intent Filter

2.4.2 节实现 Activity 跳转时用到了 Intent 类，该类的主要作用是完成通信功能，传递数据。Google 的官方解释是 Intent 负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用（比较抽象，不容易理解）。Intent 在 Android 中起到非常重要的作用，它有两个重要的属性 action 和 data，关于它的深度解析，后续章节中会给出，在此只是使用 Intent 在不同 Activity 间跳转。

Intent 一般有显式使用和隐式使用。所谓显式就是指明需要启动的组件名称，比如 Part02 项目中用于启动其他 Activity，这种应用一般只能出现在本应用程序内部。隐式使用是通过 Intent 的属性，指定目标组件应该符合的条件，筛选满足条件的组件启动。此时这些组件（包括 Android 系统组件）都可以定义一个或多个 IntentFilter（用于声明可以响应或处理的 Intent）。Part02 项目中 MainActivity 中的 IntentFilter 便指定了条件，作为应用程序的入口存在。

2.5 程序调试与应用发布

Android 提供了比较完善的程序调试功能，借助 Eclipse 的 ADT 插件，可以比较轻松地输出程序的提示信息，进行断点调试。

2.5.1 Console 与 LogCat

不同于 Java 应用程序的 Console 控制台，Android 使用 LogCat 输出日志信息。在 Eclipse 中展开 Window 菜单下的 Show View 选项，选择 Console，打开控制台，可以查看虚拟机的启动和 Android 应用程序安装提示信息，但无法查看应用程序内部的输出信息。重复刚才的操作，选择 Other 选项，打开如图 2.32 所示的窗口，选择 LogCat 就可以打开日志面板。

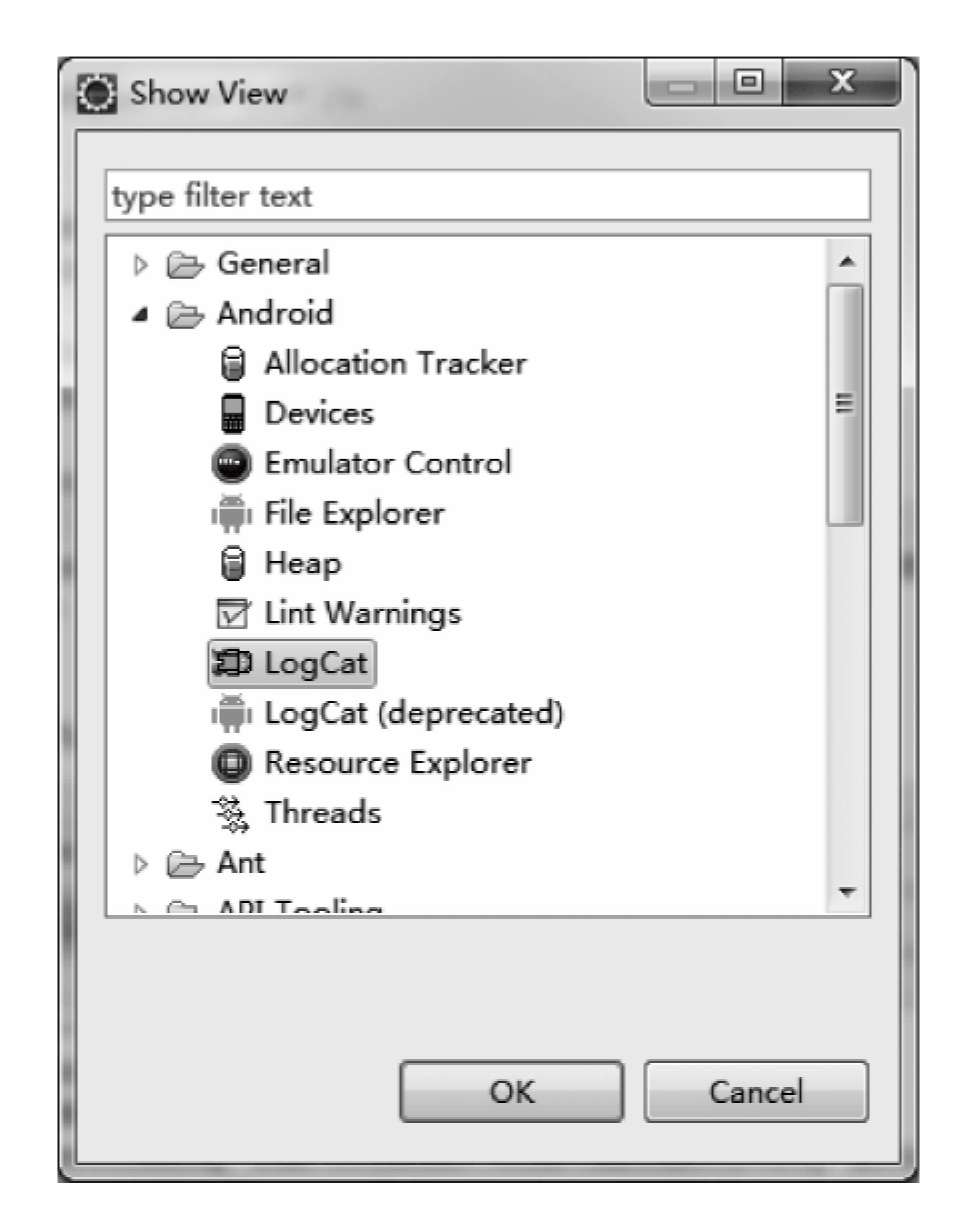


图 2.32　选择 LogCat 视图界面

开发过程中输出提示信息主要借助 LogCat，在上面测试 Activity 生命周期时，就是借助 LogCat 输出提示信息的。LogCat 控制面板如图 2.33 所示。左边是日志的过滤器，可以指定显示符合规则的日志信息。右边是日志信息，包含 5 个部分：Level(日志级别)、Time(虚拟机时间)、PID(进程号)、Application(应用程序名)、Tag(日志标题，输出日志时确定)和 Text(日志内容)。

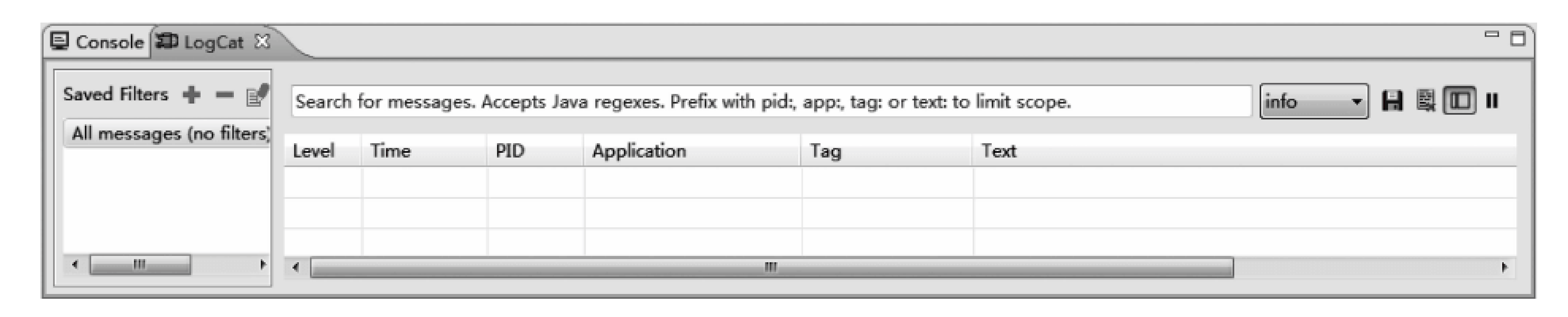


图 2.33　LogCat 控制面板

日志有优先级之分，Android 把日志按照从低到高(越致命的信息级别越高)分为 V(Verbose)，这是最低级日志，如果选择显示这个级别的日志，那就意味着显示所有日志信息；D(Debug)；I(Info)，这是前面测试时使用的一个级别；W(Warning)；E(Error)，如果出现这个日志，程序将不能正确运行；F(Fatal)；S(Silent)。在程序中输出日志信息使用 Log 类(位于 android.util 包中)、Log.v()、Log.d()、Log.i()、Log.w()和 Log.e()方法可以输出对应级别的日志信息，方法需要两个参数，分别对象 Tag 和 Text。

控制面板右侧 4 个按钮的功能依次是保存日志，清空日志内容，显示或隐藏日志过滤器，暂停或继续接收日志信息。

由于日志信息比较多，在众多内容中查找指定内容的信息可以通过 Filters。创建过滤器可以通过 LogCat 面板左边的绿色+，打开如图 2.34 所示的 Filter 创建界面。Filter

Name 是过滤器名称，将来出现在控制面板左侧的列表中。by Log Tag 是通过 Tag 标签创建过滤器，用于筛选 Tag 项的内容。by Log Message 是通过 Text 项筛选日志。by PID 是通过进程号筛选日志。by Application Name 是通过应用程序名称筛选日志。by Log Level 是通过日志的基本筛选。在 Part02 项目中，LogCat 面板中 info 过滤器（参考图 2.25～图 2.27）就是通过 Tag 过滤标题为 MainActivity 的日志信息。

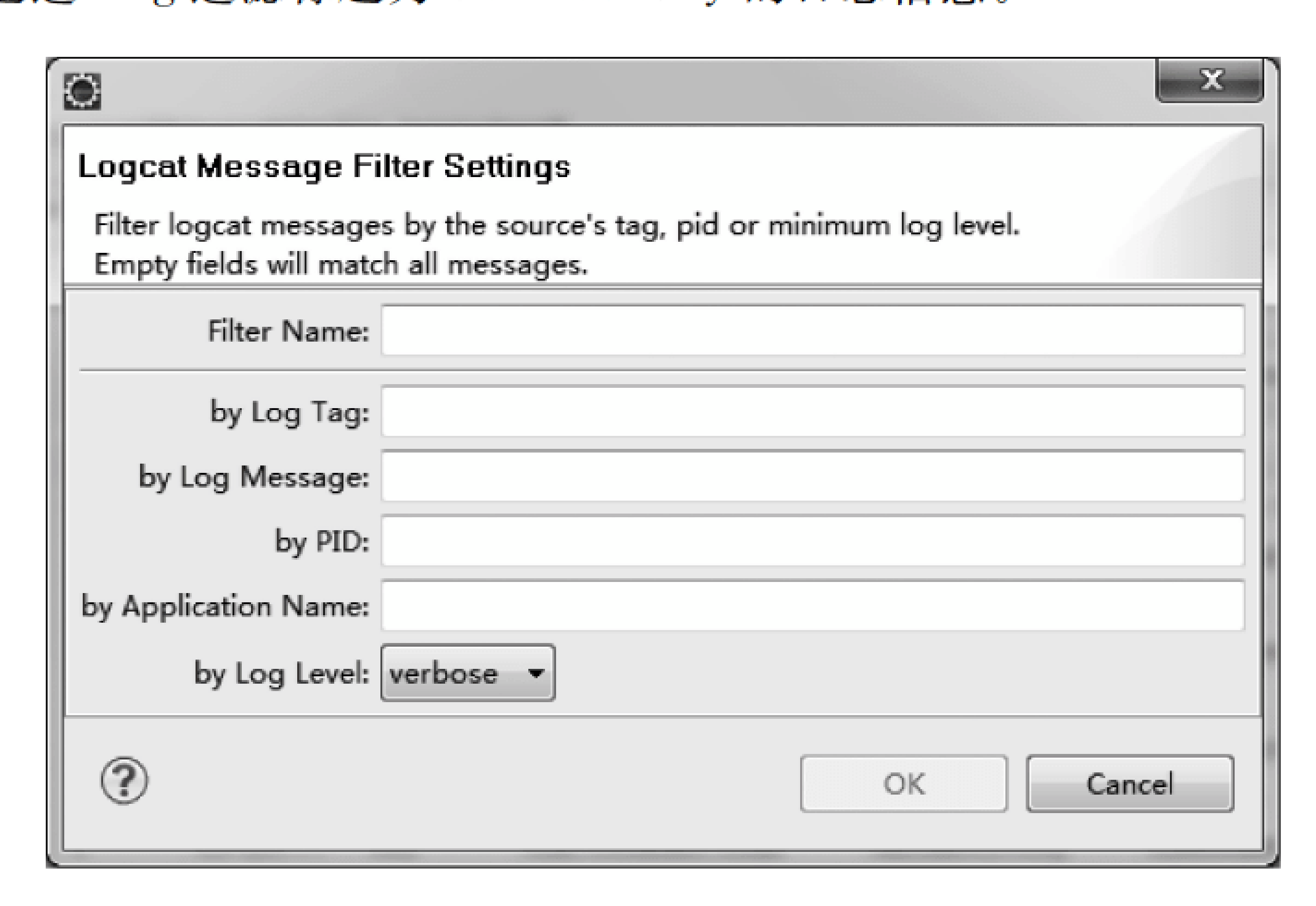


图 2.34　LogCat 过滤器设置界面

2.5.2　断点调试

和普通 Java 程序一样，Android 应用程序中支持设置断点，进行断点调试。双击代码左侧边缘就可以添加断点。在运行时选择 Debug as→Android Application 命令就可以开启调试模式。具体的代码调试方式与 Java 中一致，在此不再赘述。

2.5.3　打包发布与签名

Android 应用程序的安装包是后缀名为.apk 的文件，包含编译源代码、资源文件、版本信息等内容。如果要在手机上安装应用程序，还需要数字证书签名。这个数字证书签名就是把应用程序与开发者捆绑在一起，一方面可以保护开发者的知识产权，另一方面可以追查应用程序的开发者是谁，这与在合同上签名是一样的道理。如果想将自己开发的应用程序发布到各种 App Market，就需要进行签名。Eclipse 的 ADT 插件可以完成这个功能。

在项目名上右击，选择 Android Tools→Export singed application package 命令，打开数字签名向导。确定打包的项目，单击 Next 按钮，打开如图 2.35 所示的窗口。第一次发布程序选择 Create new keystore 单选按钮，创建新的签名文件，输入保存位置和密码，单击 Next 按钮，打开如图 2.36 所示详细信息设置界面。按照提示信息设置相应内容，完成后单击 Next 按钮，最后再确定打包生成的 apk 文件的存放位置，完成数字签名向导。如果以后再发布应用程序，选择已经存在的签名文件即可。

正式的发布过程需要按照上面的描述进行，如果只是在真机上进行安装测试，或同学朋友之间测试运行，不用打包发布也能完成这个功能。找到工作空间\项目名\bin\目录，会发现有一个项目名.apk(以项目名称命名)文件。这是 ADT 插件为了虚拟机能够顺利运行应用程序，采用调试签名进行打包发布的，该文件无法正式发布，但是可以正常在手机上安装运行。

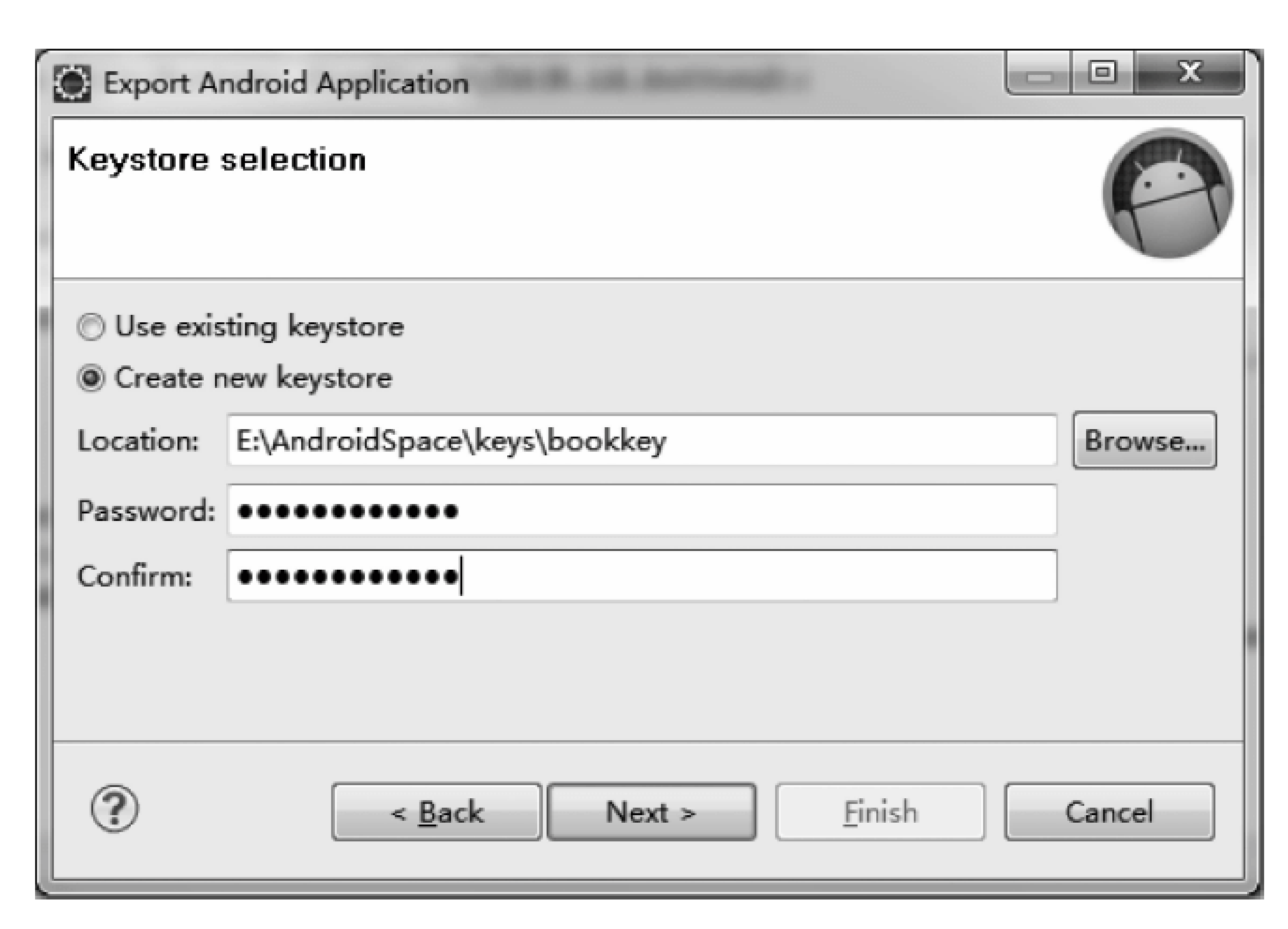


图 2.35　创建签名界面



图 2.36　详细签名信息设置界面

习　题

1. Android 操作系统从哪一个版本开始对大屏设备的 UI 设计进行了改进？

2. Activity 在 Android 应用程序中起到什么样的作用？

3. Activity 的生命周期方法有哪些？它们分别对应 Activity 的哪些状态？

4. 对一个 Activity 来说，在什么情况下会发生执行 onPause→onResume 方法的调用？

5. Android 项目的结构中不同的资源如何存放？对 res 文件夹下的文件命名有何原则？



CHAPTER 3

第3章

基本控件与布局管理器

本章主要内容：

android. wigdet 包；
Form widget 控件；
TextFields 控件；
布局管理器；
Image 和 Media 控件；
Time 和 Date 控件。

3.1 widget 包与控件

开发 Android 应用程序，设计制作能吸引用户的界面(User Interface，UI)是最重要的，UI 对于应用软件恰如外貌对于美女。界面能够影响用户的第一印象，决定了用户是否愿意体验我们开发的软件。Android 为开发 UI 提供了很多控件(组成 UI 的元素)，这些控件位于 android. widget 包中，继承自 View 类或 ViewGroup 类。继承 View 类的控件一般都是可见的，可以与使用者交互；继承 ViewGroup 类的控件一般都是不可见的布局控制器，用于存放其他的控件或布局。

Android 提供了一系列 View 类控件(如按钮、文本框、下拉列表)和 ViewGroup 布局(如线性布局、相对布局等)，熟练使用这些控件可以快速开发界面精美的 Android 应用程序。应用软件的界面基本都是由控件和布局有机组合实现，对于界面复杂的软件可以采用布局中嵌套布局的形式实现。界面设计的一般结构如图 3.1 所示。

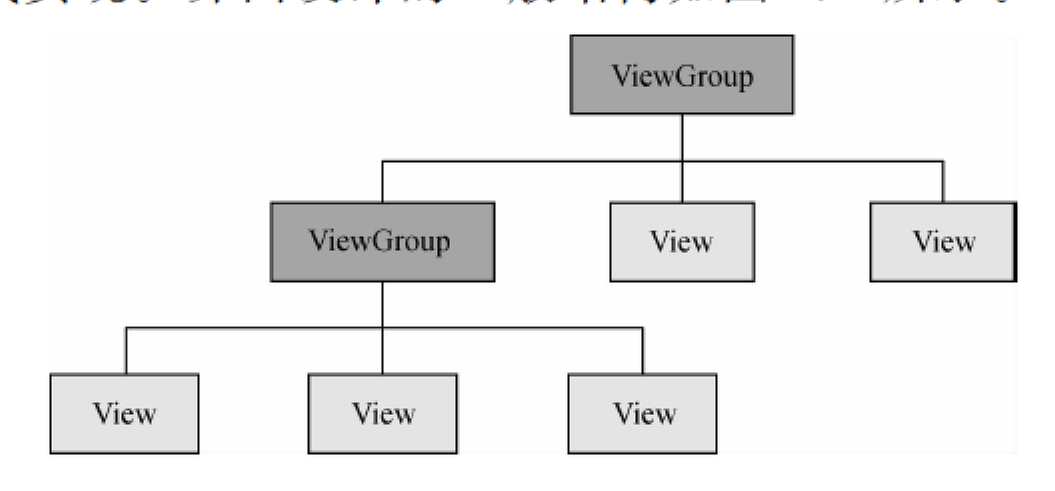


图 3.1 UI 设计的一般结构

3.1.1　控件的分类

展开 Android 项目中 res 文件夹，双击打开 layout 文件夹中的布局文件 main. xml。如果使用的 ADT 版本较高，该布局文件名称为 activity_main. xml。布局设计界面主要分为左右两个部分。左边是 Android 提供的系统控件，右边是当前界面的预览视图。界面的编辑可以通过下方的 Graphic Layout 按钮和 main. xml 按钮切换，图形化的编辑方式较为直观，操作比较简单，将选定的控件直接拖到右侧的预览视图即可，所见即所得，但是处理较为复杂的界面设计时力不从心。XML 文件编辑方式通过标签和属性定义控件，可以更加灵活地对布局进行设计，在处理复杂界面设计时一般都采用这种方法。

Graphic Layout 界面的上方是对设备屏幕的设定，包括对屏幕尺寸的设定、横竖屏的设定（Portrait 为竖屏，Landscape 为横屏）、显示的样式等。

单击 Pallete 右边的下拉三角，可以选择系统控件的预览模式。默认是 Show Small Previews，如图 3.2 所示。切换为 Show Only Icons 的效果如图 3.3 所示。

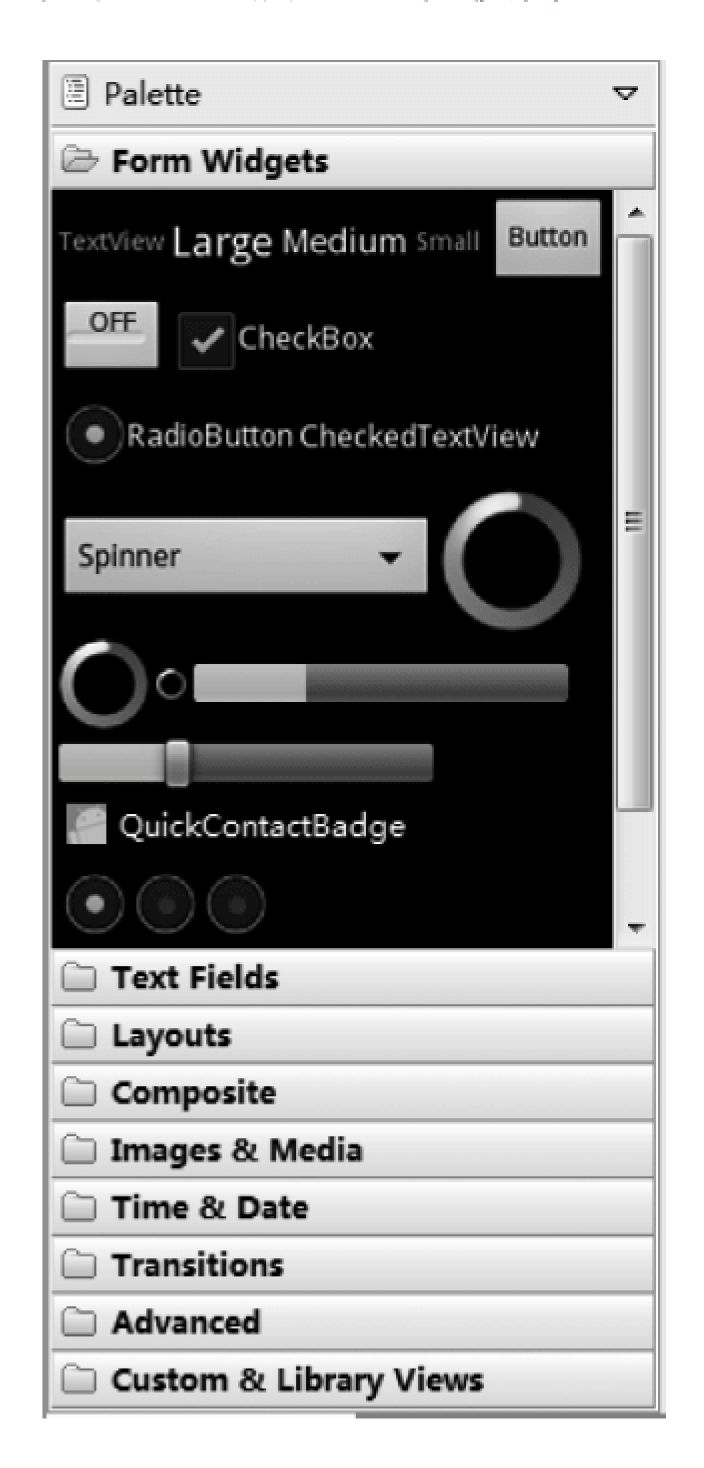




图 3.2　预览模式的控件

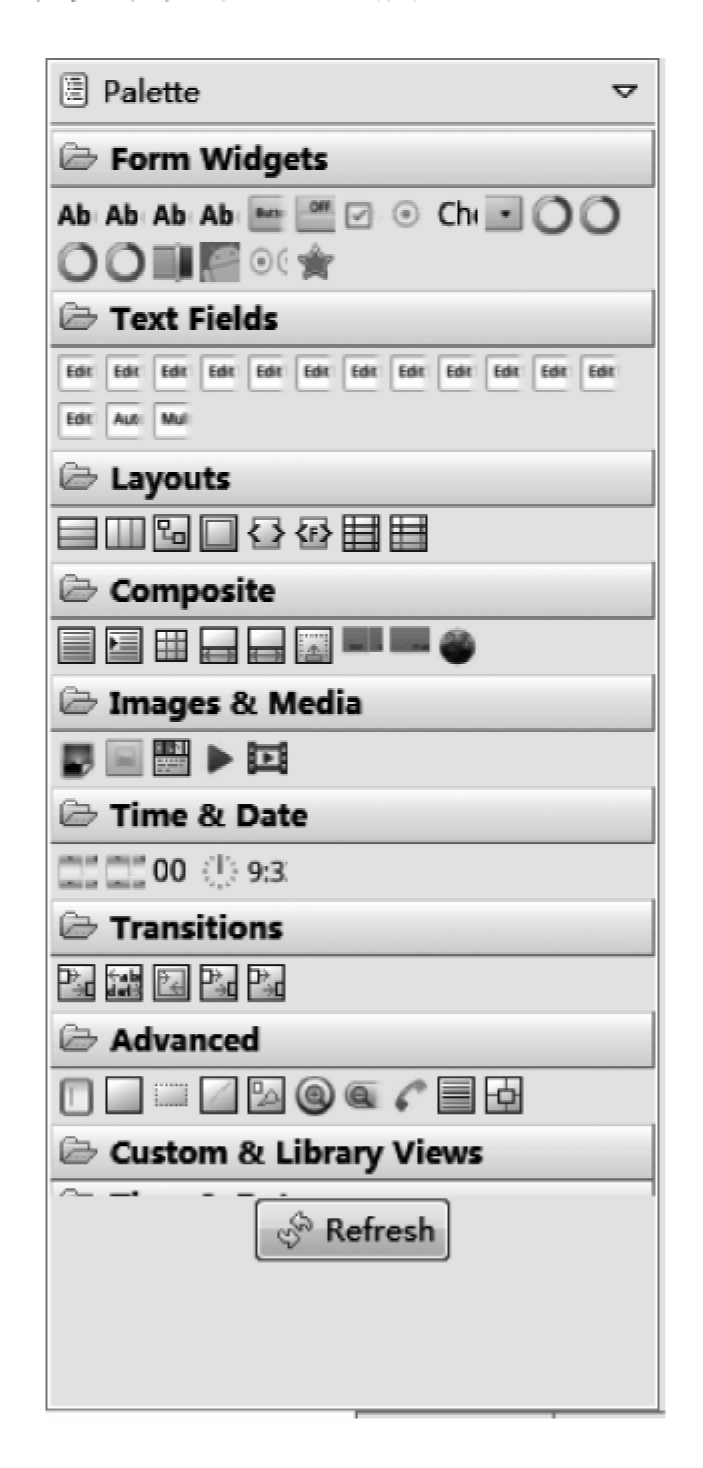


图 3.3　图标模式的控件

Android 的系统控件共分为 8 类，基本包括应用程序中所需要的大部分控件，在有特殊需要的情况下，可以自定义其他控件。

（1）Form Widgets，表单类控件包括 TextView（文本标签）、Button（按钮）、RadioButton（单选按钮）、CheckBox（复选按钮）、ProgressBar（进度条）、Spinner（下拉列表）等。

（2）Text Fields，文本框控件，根据输入内容的不同分为普通文本框、电话文本框、数字文本框等。

(3) Layouts,布局控件包括 LinearLayout(线性布局)、RelativeLayout(相对布局)、FrameLayout(单帧布局)、TableLayout(表格布局),另外再加上 AbsoluteLayout(绝对布局),组成 Android 应用开发中的 5 大布局管理器。通过这 5 种布局管理器的相互嵌套,可以实现复杂的界面布局。新版本中增加了 GridLayout(网格布局),可以更加容易地对界面进行划分。

(4) Composite,组合控件,这是比较复杂的控件,包括 ListView(列表视图)、GridView(表格视图)、ScrollView(滚动视图)、TabHost(标签容器)、WebView(webkit 内核浏览器)等。

(5) Images 和 Media,图片和媒体控件,包括 ImageView(图片视图)、ImageButton(图片按钮)、VideoView(视频视图)等控件。

(6) Time 和 Date,包括 TimePicker(时间选择器)、DatePicker(日期选择器)、AnalogClock(时钟)、DigitalClock(电子时钟)等控件。

(7) Transitions,包括 ImageSwitcher(图片切换)、ViewSwitcher(视图切换)等控件。

(8) Advanced,包括一些高级控件,使用时稍微复杂一些。

3.1.2 UI 的编辑方式

Android 应用程序中 UI 的设计主要有两种方式,一种是使用 Layout 文件设计布局,然后指定 Activity 使用该布局文件;另一种是直接继承 View 视图,使用代码绘制 Activity 的视图(游戏中主要采用这种)。借助 Eclipse 完成第一种 UI 布局比较简单。使用 Layout 布局文件充当界面时,首先要创建布局文件。

展开项目,选择 Layout 文件夹,单击右键,选择 New→Other 命令,打开"新建文件"窗口(第一次新建 Android XML 文件需要从 Other 中选择,以后可以直接在 New 中选择),如图 3.4 所示。选择 Android XML File,单击 Next 按钮,打开文件配置窗口,如图 3.5 所示。

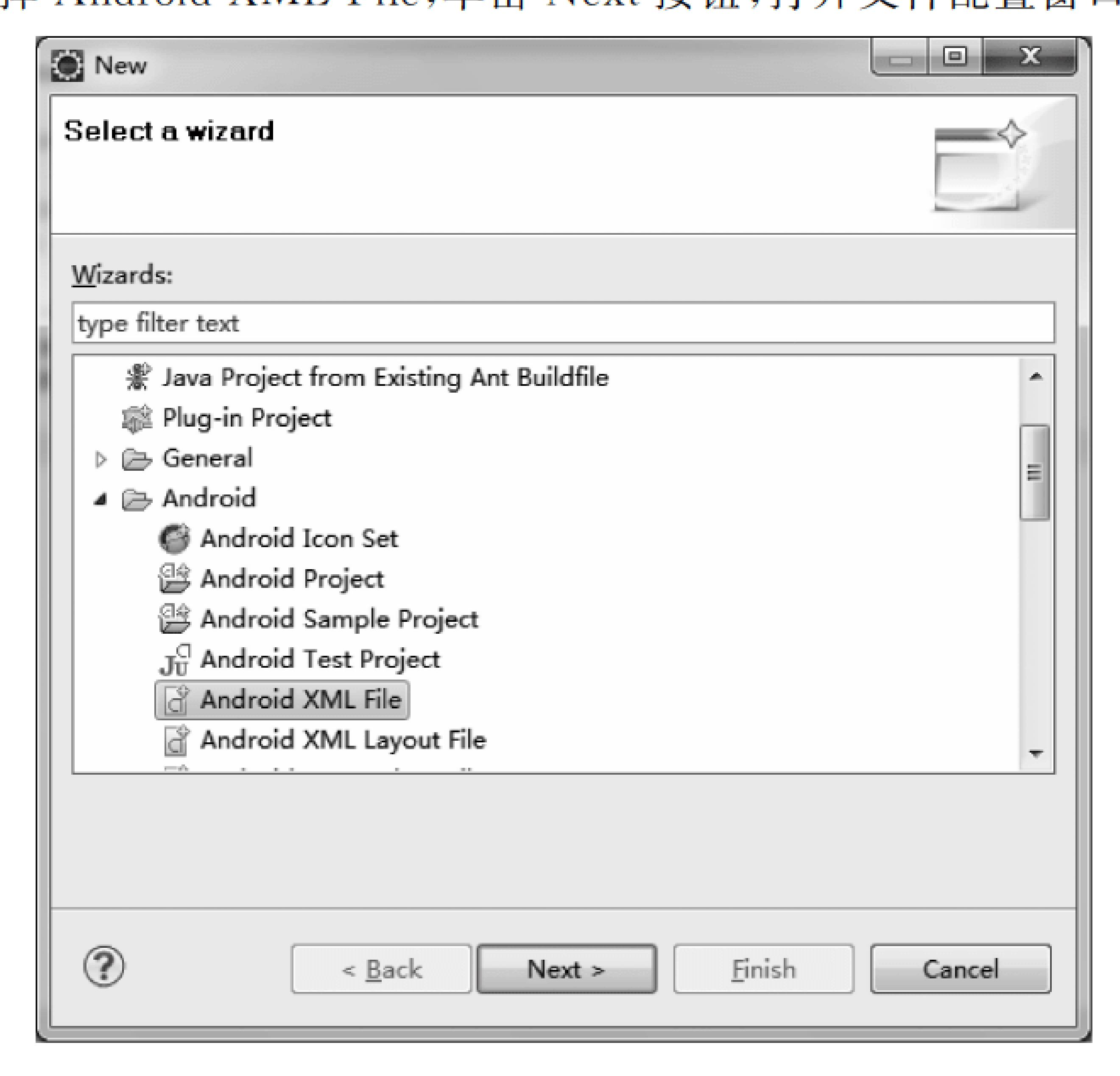


图 3.4 "新建文件"窗口



图 3.5　XML 文件配置窗口

(1) Resource Type：新建文件的类型，默认是 Layout，即布局文件，可以选择其他文件类型，如 Values，用于存放字符串数据。

(2) Project：所属项目。

(3) File：文件名，只能用小写字母、数字和_，并且以小写字母开头。

(4) Root Element：选择布局文件的布局管理器，默认选择 LinearLayout 线性布局。

确定布局之后，单击 Finish 按钮，结束文件创建过程。布局打开时默认是图形化编辑模式(Graphic Layout)。根据 UI 的设计，选择左侧恰当的控件，直接拖动到右侧的预览视图中即可。这种图形化编辑模式比较简单，但无法编辑复杂的 UI。单击底部的文件名按钮，可以切换到代码视图，所有的控件都可以使用标签直接在代码视图中创建。

3.1.3　控件的属性

控件添加到布局文件中后，可以根据需要调整相应属性。在 Graphic Layout 视图中可以在控件上单击右键，配置控件的各项属性。在代码视图中需要在标签内部指定属性名，然后赋值。Android 中控件的属性比较多，表 3.1 列出了常用的一些属性，以及它们的含义和取值信息。

表 3.1　Android 控件的常用属性

属性名	简　　介	取值信息
android:id	控件的 ID,具有唯一性	自定义
android:layout_width	控件宽度	系统值： fill_parent 填充(充满)父容器 match_parent 匹配父容器 wrap_content 包围内容 自定义值： 直接指定控件尺寸
android:layout_height	控件高度	
android:text	显示的文本信息	引用 value 中字符串,或直接指定字符串值
android:background	设定背景图片或颜色	引用 drawable 中的图片,或直接给出 RGB 颜色
android:textColor	文字颜色	
android:textSize	文字大小	
android:textStyle	文字风格	normal、bold、italic
android:maxLines	最大行数	
android:gravity	文字的对齐方式	top、bottom、left、right 等
android:password	文本输入框是否是密码	true、false
android:selectAllOnFocus	文本输入框在获得焦点时全选文字	true、false
android:inputType	文本输入框的输入内容	number、date、time 等
android:textAppearance	文字的显示大小	? android:attr/textAppearanceLarge 等系统值
android:padding	内容距控件边缘的填充间距	
android:onClick	控件单击时执行的方法	方法名
android:layout_gravity	控件在父容器中的对齐方式	top、bottom、left、right 等
android:layout_margin	距离其他控件的边缘	

以上列出的控件属性并没有区分控件,有的属性需要用于特定的控件,比如 android:inputType 属性用于指定文本输入框的类型,明确用于输入什么类型的内容。代码 03-1 是对上述属性的具体使用,该布局文件位于项目 Part03,布局文件名为 mylayout.xml,完整代码请查看随书配套资料。该段代码所设计的 UI 如图 3.6 所示。

代码 03-1

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="100px"
        android:text="文本视图控件"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:background="#FFFFFF"
```

```
            android:textColor = "#000000"
            />
        <Button
            android:id = "@ + id/button1"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "按钮 1" />

        <Button
            android:id = "@ + id/button2"
            android:layout_width = "fill_parent"
            android:layout_height = "wrap_content"
            android:text = "@string/bt2" />
        <Button
            android:id = "@ + id/button3"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "按钮 3"
            android:layout_gravity = "right"
            />
    </LinearLayout>
```

布局文件由标签构成，所有标签都需要成对出现，形如<LinearLayout></LinearLayout>，标签内部可以嵌入其他标签；形如<Botton/>，标签内部不能嵌入其他标签(如果有 HTML 基础，对这种编辑方式会感到亲切)。每个标签(某种控件)的属性写在标签的开始部分，属性结尾没有(这种属性赋值方式与 CSS 类似)。

图 3.6 中使用了 4 种标签完成布局。最外层是 LinearLayout 标签，这是线性布局管理器，对于放入该标签内部的控件按照线性排列。属性 android:layout_width 和 android:layout_height 指定线性布局管理器的宽和高，值 match_parent 指明宽高需要匹配父容器(此处父容器指手机显示屏区域)。android:orientation 属性对于线性布局非常重要，指明是水平线性(取值 horizontal)或是竖直线性(取值 vertical)。



图 3.6 mylayout 布局界面

LinearLayout 标签的内部是一个 TextView 标签和三个 Button 标签。对于控件一般需要指明其 android:id 属性(在需要的情况下，布局管理器也可以指明 id 属性)。android:id 属性的赋值需要使用@+id/id 名(id 名是自定义的)，其中+表示当修改完某个布局文件并保存后，系统会自动在 R.java 文件中生成相应的 int 类型变量，以方便在代码中采用R.id.id 名的方式引用该控件。例如，@+id/button1 会在 R.java 文件中生成int button1=value 的属性，其中 value 是一个十六进制的数，是自动计算产生的。

如果需要在布局文件中引用其他资源，需要使用@id/id 名(引用其他控件)、@string/字符串名(引用 values 文件夹下 strings 中字符串)、@drawable/图片名(引用 drawable-×××文

件下中图片)或@layout/布局名(引用其他布局)的方式实现。比如 android:text="@string/bt2",其值是引用 strings.xml 文件中名为 bt2 的字符串。

Android 控件的属性比较多,这些属性大多来自 View 类。另外,不同控件又具有特殊属性,在使用过程中需要注意以下几点。

- 先了解一般常用属性及其作用,切勿贪多。
- 掌握 android:id 属性的使用,包括如何增加 id 和引用其他资源。
- 控件都需要设置宽高属性。
- 控件的赋值建议采用引用赋值方式,这样便于后期的国际化,不过为了呈现更加直观的效果,在后续讲解中大多采用了直接赋值的方式。

3.2 Form Widgets

Form Widget 列表中包含 TextView、Button、ToggleButton、RadioButton、CheckBox、CheckedTextView、ProgressBar、SeekBar、Spinner、QuickContactBadge、RadioGroup 和 RatingBar。下面对其中比较重要的控件逐一介绍。

3.2.1 TextView

用于显示字符串的文本标签,不能编辑。TextView 的属性 android:textAppearance 规定文字的显示方式,可以使用如下系统值,效果如图 3.7 所示。

(1) ? android:attr/textAppearanceLarge,大字号显示文字。

(2) ? android:attr/textAppearanceMedium,中等字号显示文字。

(3) ? android:attr/textAppearanceSmall,小字号显示文字。

(4) 默认,默认字号显示文字。



图 3.7 TextView 不同字号的显示

3.2.2 Button

Button 控件几乎是家喻户晓了,主要用于单击操作,处理相应事件。在 Android 中按钮的事件处理方式有两种,一种是直接给按钮注册监听器(这种方式与 J2SE 中事件处理模型一致),另一种是 android:onClick 属性,直接指定处理单击事件的方法。

代码 03-2 演示两种事件处理方式,完整代码请查看随书配套资料 Part03 项目,MainActivity.java 源代码文件。在 onCreate 方法中,初始化三种控件,通过 findViewById(通过 Id 获取 View 控件)方法,获取布局文件 layoutbutton 中的三个控件,并强转为对于类型。b1 按钮直接注册监听器(采用匿名内部类实现)OnClickListener(注意此监听器是 android.view.View.OnClickListener,不要引错),监听按钮的单击。b2 按钮没有注册监听

器，在布局文件（如代码 03-3 所示）中，采用 android:onClick="buttonClick"属性，指明由方法 buttonClick 处理该按钮的单击事件。

代码 03-2

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layoutbutton);
        //实例化控件
        b1 = (Button) findViewById(R.id.bt_1);
        b2 = (Button) findViewById(R.id.bt_2);
        tv = (TextView) findViewById(R.id.tv_1);
        //注册监听器,监听器由匿名内部类实现
        b1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                tv.setText("监听器处理按钮单击!");
            }
        });
    }
    //处理按钮单击事件
    public void buttonClick(View v){
        tv.setText("方法处理按钮单击!");
    }
```

定义 buttonClick 方法时，有两点需要注意。一是此类方法必须是 public 修饰，二是参数列表只能有一个 View 类型参数。当指定按钮被单击时，作为 View 传入此方法。

代码 03-3

```
<TextView
        android:id="@+id/tv_1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textAppearance="?android:attr/textAppearanceLarge"
        />
    <Button
        android:id="@+id/bt_1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="注册监听器响应单击" />
    <Button
        android:id="@+id/bt_2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="指明响应单击的方法"
        android:onClick="buttonClick"
        />
```

3.2.3 ToggleButton

ToggleButton(开关按钮)是Android系统中比较简单的一个组件,是一个具有选中和未选择双状态的按钮,并且需要为不同的状态设置不同的显示文本。常用属性如下。

(1) android:textOn="开启",当按钮处于选中状态显示的文字。

(2) android:textOff="关闭",当按钮处于未选中状态显示的文字。

(3) android:disabledAlpha="0.1",当按钮未选中时,按钮的Alpha值,取值范围为0~1。

ToggleButton常用的监听器是OnClickListener和OnCheckedChangeListener,都可以监听按钮的状态变化。代码03-4演示了两种监听器处理ToggleButton的状态改变,完整代码参考随书配套资料Part03项目,MainActivity.java文件,布局文件是layouttogglebutton.xml。

代码 03-4

```
ToggleButton tb1,tb2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //测试 toggleButton
        setContentView(R.layout.layouttogglebutton);
        //初始化控件
        tb1 = (ToggleButton) findViewById(R.id.toggleButton1);
        tb2 = (ToggleButton) findViewById(R.id.toggleButton2);
        //注册监听器
        tb1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                if(tb1.isChecked()){
                    setTitle("tb1 On!"); //设置应用程序的标题
                }else{
                    setTitle("tb1 Off!"); //设置应用程序的标题
                }
            }}
        );
        tb2.setOnCheckedChangeListener(new OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                    boolean isChecked) {
                if(isChecked){
                    setTitle("tb2 开启!");
                }else{
                    setTitle("tb2 关闭!");
                }
            }
        });
    }
```

在OnClickListener监听器中,通过ToggleButton的isChecked方法,判断按钮是否选

中。在 OnCheckedChangeListener 监听器中，可以通过 onCheckedChange 方法的第二个参数判断按钮是否处于选中状态。运行效果如图 3.8 所示。

ToggleButton 是 CompoundButton 类的子类。CompoundButton 继承 Button 类，具有两种状态：选中或未选中，按钮被单击时状态会发生切换，该类按钮常常被用于功能开关，如开启背景音乐或关闭背景音乐。CompoundButton 类的另外两个常用子类是 RadioButton 和 CheckBox。它们都比 Button 类多一个监听器，即 OnCheckedChangeListener。



图 3.8　ToggleButton 运行效果

3.2.4　RadioButton 与 RadioGroup

RadioButton 是单选按钮，继承自 CompoundButton。RadioGroup 是单选按钮组，继承自 LinearLayout 类，本身不能直接使用，需要与 RadioButton 配合使用，用于将 RadioButton 聚合成一组。

代码 03-5 演示如何使用 RadioGroup，完整代码请参考随书配套资料（这句话出现了很多次，目的是初学者不要盲目照抄下面的代码，然后就理所当然地运行，那样肯定是无法运行的）Part03 项目，MainActivity.java 文件，布局文件是 layoutradiogroup.xml。

代码 03-5

```
RadioGroup rgroup;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //测试 RadioButton 与 RadioGroup
    setContentView(R.layout.layoutradiogroup);
    tv = (TextView) findViewById(R.id.tv_rg);
    rgroup = (RadioGroup) findViewById(R.id.radioGroup1);
    rgroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            tv.setText("当前选中控件 Id: " + checkedId);
            RadioButton rb = (RadioButton) findViewById(checkedId);
            tv.append("\n" + rb.getText());
        }}
    );
}
```

虽然每个 RadioButton 都有 OnClickListener 监听器和 OnCheckedChangeListener 监听器，但直接处理每个单选按钮比较麻烦，需要每个都做判断，所以一般情况下可以给 RadioGroup 添加监听器，用于判断该组中哪个单选按钮被选中。

仔细阅读代码 03-5，会发现 RadioGroup 的监听器也叫 OnCheckedChangeListener，但在实现该监听器时，又添加了 RadioGroup.，这说明此处的 OnCheckedChangeListener 监听器是 RadioGroup 类中一个内部接口，为了避免产生歧义，在创建该接口的实现类时直接指

定了该接口所属的外部类，采用形如“外部类.内部接口”的方式创建匿名内部类。这种创建接口的方式在以后会经常见到。监听器中方法 onCheckedChanged 的第二个参数即为备选按钮的 ID。代码运行效果如图 3.9 所示。

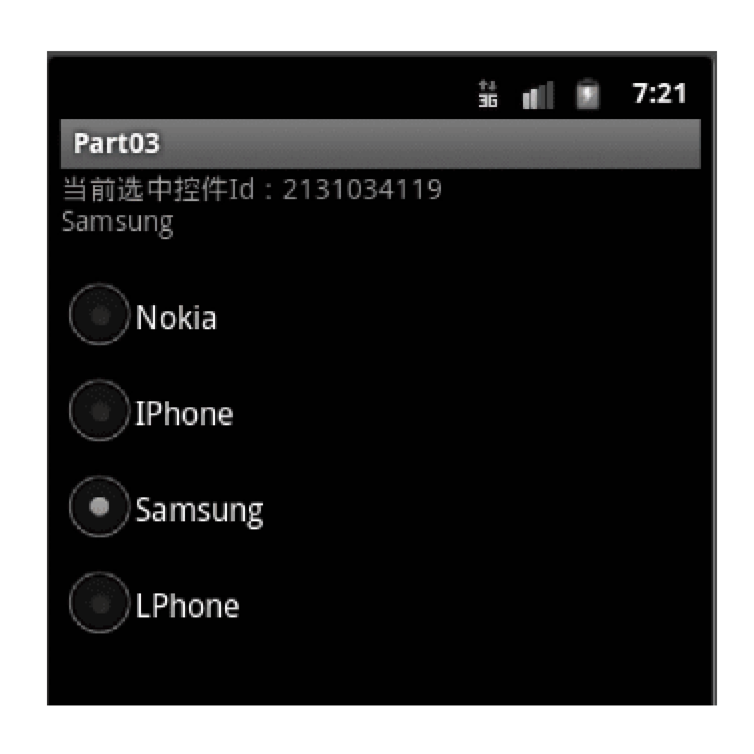


图 3.9 RadioGroup 运行效果

3.2.5 CheckBox

复选框，继承自 CompoundButton 类，具有 OnClickListener 监听器和 OnCheckedChangeListener 监听器，通过 isChecked 方法，判断该按钮是否处于选中状态。

3.2.6 CheckedTextView

该类继承自 TextView，实现了 Checkable 接口，常用于 ListView 中。使用自定义适配器，根据指定数据与布局样式填充相应数据。在后续章节学习完 ListView 与适配器后，感兴趣的读者可以尝试 CheckedTextView 与 ListView 的配合使用。

3.2.7 ProgressBar

Android 中的进度条有两类：默认的环形进度条和需要设置的水平进度条。进度条的大小和状态可以通过 style 属性设置。style 的设置信息如下。

(1) style="? android:attr/progressBarStyleLarge"，大尺寸的环形进度条。

(2) style="? android:attr/progressBarStyleSmall"，小尺寸的环形进度条。

(3) 不设置，模式尺寸的环形进度条。

(4) style="? android:attr/progressBarStyleHorizontal"，水平进度条。

或者采用如下的方式设置进度条样式。

(1) style="@android:style/Widget.ProgressBar.Small"

(2) style="@android:style/Widget.ProgressBar.Lager"

(3) style="@android:style/Widget.ProgressBar.Horizontal"

进度条常用于执行时间较长的代码块中，比如下载、更新等操作，可以给用户心理上的安慰。但是在 Android 中进度条的操作稍微麻烦一点，因为 Android 不允许子线程修改 Activity 的界面，而进度条都是由子线程控制的。Android 给出的解决办法是使用 Handler 类，处理子线程中需要更新 UI 的操作。关于 Handler 的深度介绍请阅读后续章节。

对于水平进度条，经常使用的属性有：

(1) android:max="100"，设置进度条的最大值。

(2) android:progress=10，设置第一层进度条的初始值。

(3) android:secondaryProgress="80"，设置第二层进度条的初始值。

进度条的常用方法如下。

(1) int getMax()：返回这个进度条的最大值。

(2) int getProgress()：返回当前进度。

(3) int getSecondaryProgress()：返回当前次要进度。

(4) void incrementProgressBy(int diff)：指定增加的进度，每次推进的步伐。

(5) boolean isIndeterminate()：指示进度条是否在不确定模式下。

(6) void setIndeterminate(boolean indeterminate)：设置不确定模式下，用于无法确定时间的任务。

(7) void setVisibility(int v)：设置该进度条是否可视。

代码 03-6 演示如何使用 Handler 类在子进程中更新 UI，完整代码请参考随书配套资料 Part03 项目，MainActivity.java 文件，布局文件是 layoutprogressbar.xml。

代码 03-6

```
ProgressBar pb1,pb2;
Handler handler = new Handler();
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //测试进度条
    setContentView(R.layout.layoutprogressbar);
    pb1 = (ProgressBar) findViewById(R.id.progressBar1);
    pb2 = (ProgressBar) findViewById(R.id.progressBar4);
    changeProgress();
}
private void changeProgress() {
    Thread t = new Thread(new MyThread());
    t.start();
}
class MyThread implements Runnable{
    @Override
    public void run() {
        while(pb2.getProgress()<pb2.getMax()){
            handler.post(new Runnable(){
                @Override
                public void run() {
                    pb2.setProgress(pb2.getProgress() + 2);
                    pb2.setSecondaryProgress(pb2.getSecondaryProgress() + 10);
                    if(pb2.getSecondaryProgress()>= pb2.getMax())pb2.setSecondaryProgress(0);
                }
            });
```

```
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
```

上述代码将修改进度条的功能放在 handler.post(匿名线程)中完成,由 handler 交给(post)Activity UI 的主线程更新进度。三个环形进度条没有处理进度,都是默认为 indeterminate 模式,在这种模式下,进度条将一直循环,等延时代码执行结束时,将进度条隐藏即可。代码运行效果如图 3.10 所示。

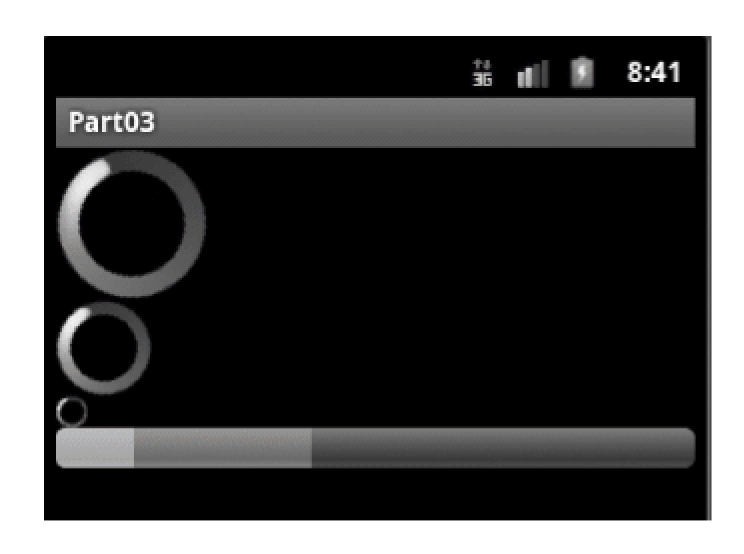


图 3.10 ProgressBar 运行效果

在 Android 中使用子线程需要注意以下事项。

- 一个 Android 应用程序的 UI 由一个主线程维护,不允许子线程修改。
- 子线程不能直接更新 Activity 中的 UI,需要使用 Handler 通知主线程修改 UI。
- Handler 的详细解释在后续章节,此处暂不深究。

3.2.8 SeekBar

拖动条是进度条的间接子类,拥有进度条的所有属性和方法,支持滑块的推动。代码 03-7 演示如何使用 SeekBar,完整代码请参考随书配套资料 Part03 项目,MainActivity.java 文件,布局是 layoutseekbar.xml。

📖 代码 03-7

```
SeekBar sbar;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试拖动条
    setContentView(R.layout.layoutseekbar);
   tv = (TextView) findViewById(R.id.tv_sb);
   sbar = (SeekBar) findViewById(R.id.seekBar1);
   //sbar 注册监听器
```

```
sbar.setOnSeekBarChangeListener(new OnSeekBarChangeListener(){
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
            boolean fromUser) {
        tv.setText("当前进度: " + progress);
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {

    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {

    }}
);
```

SeekBar 拥有 OnSeekBarChangeListener 监听器，当滑块滑动时执行监听器中的对应方法。onStartTrackingTouch 方法和 onStopTrackingTouch 方法对应滑动的开始和结束，onProgressChanged 方法对应进度值的改变。属性 android:thumb="@drawable/sb0"可以设定滑块为指定图片，该图片最好是 png 格式(支持透明效果)。代码运行效果如图 3.11 所示。



图 3.11　SeekBar 运行效果

3.2.9　Spinner

Spinner 是类似于下拉列表的一种控件，用户可以从中选择相应选项。Spinner 中的数据需要使用 Adapter(适配器)填充，适配器在后续章节中介绍。代码 03-8 演示如何使用 Spinner 控件，完整代码请参考 Part03 项目，MainActivity.java 文件，布局是 layoutspinner.xml。

代码 03-8

```
Spinner spinner;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 Spinner
    setContentView(R.layout.layoutspinner);
    spinner = (Spinner) findViewById(R.id.spinner1);
    tv = (TextView) findViewById(R.id.tv_spinner);
    spinner.setPrompt("请选择三国人物: "); //
    final List list = new ArrayList();
    list.add("郭嘉");
    list.add("诸葛亮");
    list.add("周瑜");
```

```
        list.add("司马懿");
        list.add("陆逊");
        ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_spinner_item,
    list);
        spinner.setAdapter(adapter);
        spinner.setOnItemSelectedListener(new OnItemSelectedListener(){
            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1,
                    int arg2, long arg3) {
                tv.setText("已选择: " + arg2 + " : " + arg3);
                tv.append("\n" + list.get(arg2));
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0) {

            }}
        );
```

上述代码采用 ArrayAdapter 适配器，将数据填充到 Spinner 中，并添加了 OnItemSelectedListener 监听器，处理下拉列表中元素的选择。代码运行效果如图 3.12 所示。



图 3.12 Spinner 运行效果

3.2.10 QuickContactBadge

快速联系人标记，该控件用于快速查找与指定条件匹配的联系人信息。需要开启访问联系人的权限，在 AndroidManifest.xml 文件中增加权限：

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

代码 03-9 演示如何使用 QuickContactBadge 控件，完整代码参考 Part03 项目，MainActivity.java 文件，布局文件 layoutquickcontactbadge.xml，应用程序配置文件需要增加读取联系人的权限。

📖 **代码 03-9**

```
QuickContactBadge qcb;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //测试 QuickContactBadge
    setContentView(R.layout.layoutquickcontactbadge);
    qcb = (QuickContactBadge) findViewById(R.id.quickContactBadge1);
    qcb.assignContactFromEmail("test@163.com", true);
    qcb.assignContactFromPhone("13902208866", true);
    qcb.setMode(QuickContact.MODE_SMALL);
}
```

该段代码如果要运行成功，需要在虚拟机中新建联系人信息，指明电话号码 1390220××××，邮件 test@163.com。代码运行效果如图 3.13 所示。

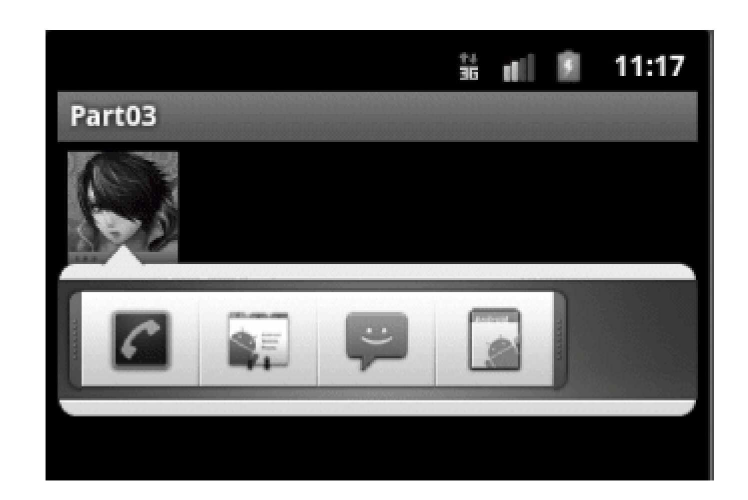


图 3.13 QuickContactBadge 运行效果

3.2.11 RatingBar

RatingBar 是基于 SeekBar 和 ProgressBar 的扩展，用星形来显示等级评定。使用 RatingBar 的默认尺寸时，用户可以触摸/拖动来设置评分。另外两种样式是小尺寸的 ratingBarStyleSmall 和大尺寸的 ratingBarStyleIndicator，其中大尺寸时只适合指示，不能用于用户交互。

（1）style＝"? android:attr/ratingBarStyleSmall"，指定为小尺寸 RatingBar。

（2）style＝"? android:attr/ratingBarStyleIndicator"，指定为大尺寸 RatingBar。

代码 03-10 演示 RatingBar 的使用，完整代码参考 Part03 项目，MainActivity.java 文件，布局是 layoutratingbar.xml。

📖 **代码 03-10**

```
RatingBar rb1,rb2,rb3;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 RatingBar
    setContentView(R.layout.layoutratingbar);
```

```
        rb1 = (RatingBar) findViewById(R.id.ratingBar1);
        rb2 = (RatingBar) findViewById(R.id.ratingBar2);
        rb3 = (RatingBar) findViewById(R.id.ratingBar3);
        tv = (TextView) findViewById(R.id.tv_rb);
        b1 = (Button) findViewById(R.id.bt_rb);
        b1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                tv.setText("评价结果: ");
                tv.append("\n商品质量: " + rb1.getProgress());
                tv.append("\n服务态度: " + rb2.getProgress());
                tv.append("\n物流速度: " + rb3.getProgress());
            }
        });
    }
```

RatingBar 的默认最大值是 10,可以通过 android:max="100"属性修改。默认尺寸的 RatingBar 可以与用户交互,支持单击或拖动。代码运行效果如图 3.14 所示。



图 3.14 RatingBar 运行效果

3.3 TextFields

在 Graphic Layout 视图中展开 Text Fields 列表,共有 15 个文本输入框,用于完成不同类型文本内容的输入,如 Plain Text(普通文本框)、Password(密码输入框)、E-mail(电子邮箱输入框)、Postal Address(通信地址输入框)、Number(数字输入框)等。这些输入框都对应 Android Widget 包中的 EditText 控件,上述诸多输入类型的划分由该控件的属性 android:inputType 指定。该属性的取值可以影响虚拟键盘的输入模式,比如作为数字输入框时,虚拟键盘会自动切换为数字输入模式,所以该属性对 EditText 控件尤为重要,可以在用户输入时减少切换输入法的操作。android: inputType 属性常用取值与说明见表 3.2。

表 3.2 inputType 属性常用取值与说明

android：inputType 取值	说明	取值类型
android:inputType="number"	输入内容为数字，虚拟键盘为切换为数字输入模式	数值
android:inputType="numberDecimal"	输入内容为带小数点的浮点格式	
android:inputType="phone"	输入内容为电话号码，虚拟键盘切换为拨号键盘	
android:inputType="datetime"	输入内容为时间日期	
android:inputType="date"	输入内容为日期键盘	
android:inputType="textCapWords"	输入文本首字母大写	字符串
android:inputType="textCapSentences"	输入文本时仅第一个字母大写	
android:inputType="textAutoCorrect"	输入文本时自动更正	
android:inputType="textAutoComplete"	输入文本时自动完成	
android:inputType="textMultiLine"	可以多行输入	
android:inputType="textUri"	输入内容为网址，虚拟键盘会显示 www、com 等内容	
android:inputType="textEmailAddress"	输入内容电子邮件地址，虚拟键盘会显示@符号	
android:inputType="textPostalAddress"	输入内容为地址	
android:inputType="textPassword"	密码输入框	
android:inputType="textVisiblePassword"	可见密码输入框	

EditText 控件是 TextView 控件的子类，是具有输入编辑功能的文本框。除了 inputType 属性，常用的属性还有 android:digits、android:editable、android:lines 等，详细描述请参考表 3.3。

表 3.3 EditText 控件的常用属性

属性	说　　明
android:digits	设置 EditText 所接受的字符，如 1234567890
android:editable	设置 EditText 是否可以编辑，取值为 true 或 false
android:gravity	文本的对齐方式
android:hint	文本框没有输入内容时显示的提示信息
android:maxLength	设置文本最大长度
android:lines	设置文本框显示的行数
android:singleLine	设置是否单行
android:textSize	文字大小，单位建议采用 sp

3.4 布局管理器

Android 布局管理器都继承自 ViewGroup 类，用于存放其他控件或嵌套其他布局。常用的布局管理器有 5 个，分别是 LinearLayout（线性布局）、RelativeLayout（相对布局）、TableLayout（表格布局）、FrameLayout（帧布局）和 AbsoluteLayout（绝对布局）。

3.4.1 LinearLayout

线性布局管理器对存放其中的控件或布局采用线性方式管理，通过属性 android:

orientation 设定线性的方向，取值 vertical 为竖直方向线性，取值 horizontal 为水平线性。线性布局的 android：gravity 属性可以设定其中元素的对齐方式。

代码 03-11 演示如何使用线性布局设计登录 UI 界面，完整代码请参考项目 Part03，布局文件 testlinearlayout.xml。运行效果如图 3.15 所示。

代码 03-11

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        >
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="账号："
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <EditText
            android:id="@+id/editText1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1">
            <requestFocus />
        </EditText>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="密码："
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <EditText
            android:id="@+id/editText2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="textPassword" />
    </LinearLayout>
    <LinearLayout
        android:id="@+id/linearLayout2"
```

```
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:gravity = "right"
         >

        <Button
            android:id = "@ + id/button1"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "登录" />
        <Button
            android:id = "@ + id/button2"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "注册" />
    </LinearLayout >
</LinearLayout >
```

上述代码中使用 LinearLayout 竖直线性布局管理器，嵌入三个水平线性布局管理器，并将最后一个线性布局管理器的 android:gravity 属性设置为右对齐。

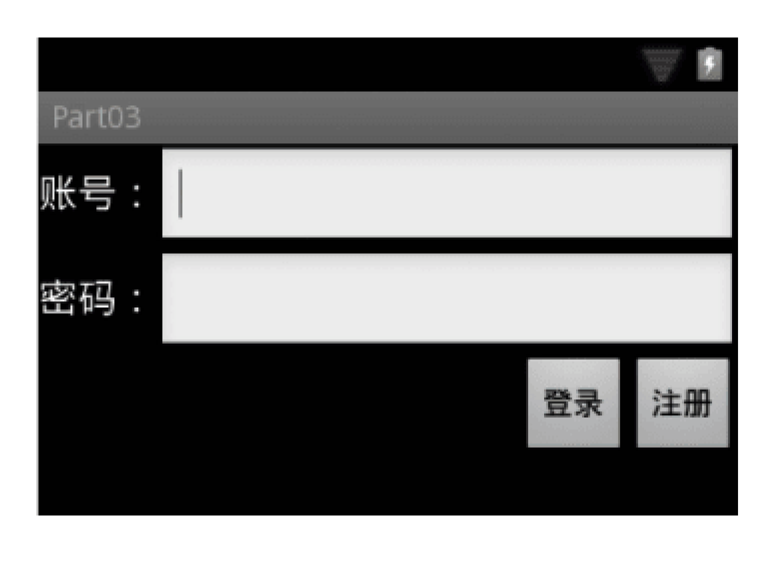


图 3.15　线性布局界面效果图

3.4.2　RelativeLayout

相对布局管理器使用元素的相对参考位置布局，在容器中的子元素可以使用彼此之间的相对位置或者与容器之间的相对位置进行定位，比如在哪个元素的哪个方位，与哪个元素采取何种对齐方式。RelativeLayout 中常用的属性有三类，第一类是属性值必须为 id 引用；第二类是属性值为 true 或 false；第三类是属性值为具体像素值，具体解释参考表 3.4。

表 3.4　RelativeLayout 常用属性介绍

属　　性	作　　用	备　　注
android:layout_below	在某元素的下方	属性值为其他控件的 id 引用，形如 @ id/id-name
android:layout_above	在某元素的上方	
android:layout_toLeftOf	在某元素的左边	
android:layout_toRightOf	在某元素的右边	
android:layout_alignTop	本元素的上边缘和某元素的上边缘对齐	
android:layout_alignLeft	本元素的左边缘和某元素的左边缘对齐	
android:layout_alignBottom	本元素的下边缘和某元素的下边缘对齐	
android:layout_alignRight	本元素的右边缘和某元素的右边缘对齐	

续表

属　　性	作　　用	备　　注
android:layout_centerHorizontal	水平居中	取值为 true 或 false
android:layout_centerVertical	竖直居中	
android:layout_centerInparent	相对于父元素完全居中	
android:layout_alignParentBottom	贴紧父元素的下边缘	
android:layout_alignParentLeft	贴紧父元素的左边缘	
android:layout_alignParentRight	贴紧父元素的右边缘	
android:layout_alignParentTop	贴紧父元素的上边缘	
android:layout_marginBottom	底边缘的距离	取值为具体的像素值，如 5px
android:layout_marginLeft	左边缘的距离	
android:layout_marginRight	右边缘的距离	
android:layout_marginTop	上边缘的距离	

代码 03-12 演示 RelativeLayout 的使用，完整代码请参考 Part03 项目，testrelativelayout. xml 布局文件，代码的布局效果如图 3.15 所示。

代码 03-12

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/r_tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="账号:"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText
        android:id="@+id/r_et1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/r_tv1"
        android:hint="请输入账号"
        >
    </EditText>
    <TextView
        android:id="@+id/r_tv2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
       android:layout_below="@id/r_et1"
        android:text="密码:"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText
```

```
            android:id="@+id/r_et2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/r_tv2"
            android:layout_below="@id/r_et1"
            android:hint="请输入密码"
            >
        </EditText>
        <Button
            android:id="@+id/r_bt1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="注册"
            android:layout_below="@id/r_et2"
            android:layout_alignParentRight="true"
            />
        <Button
            android:id="@+id/r_bt2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="登录"
            android:layout_below="@id/r_et2"
            android:layout_toLeftOf="@id/r_bt1"
            />
    </RelativeLayout>
```

在相对布局中指明元素位置时是以其他元素为参考，需要引用其他控件，可以通过@id/id name的方式实现。相对布局中的控件都应该指明id值，便于其他元素的引用。此外，由于参考控件的不同，相同的界面布局，布局代码也不尽相同。

3.4.3　TableLayout

表格布局管理器以行和列的形式对控件进行管理，每一行对应一个TableRow对象或一个View控件。当行为TableRow对象时，可以添加子控件，默认情况下，每个子控件占据一列。当行为View时，该View将独占一行。

TableLayout列数由子元素最多的TableRow确定，元素的个数即为列数。元素的宽高由表格决定，layout_width和layout_height失去原来的作用。常用属性与作用描述见表3.5。

表3.5　TableLayout常用属性与作用

属　　性	作　　用
android:stretchColumns	设置可伸展的列，伸展后使得该行元素充满整行
android:shrinkColumns	设置可收缩的列，当内容没有占满一行时没有效果
android:collapseColumns	设置可隐藏的列
android:layout_column	指定单元格显示在第几列
android:layout_span	指定单元格跨几列

代码 03-13 演示 TableLayout 的使用，完整代码参考 Part03 项目，testtablelayout.xml 布局文件，代码运行效果如图 3.16 所示。

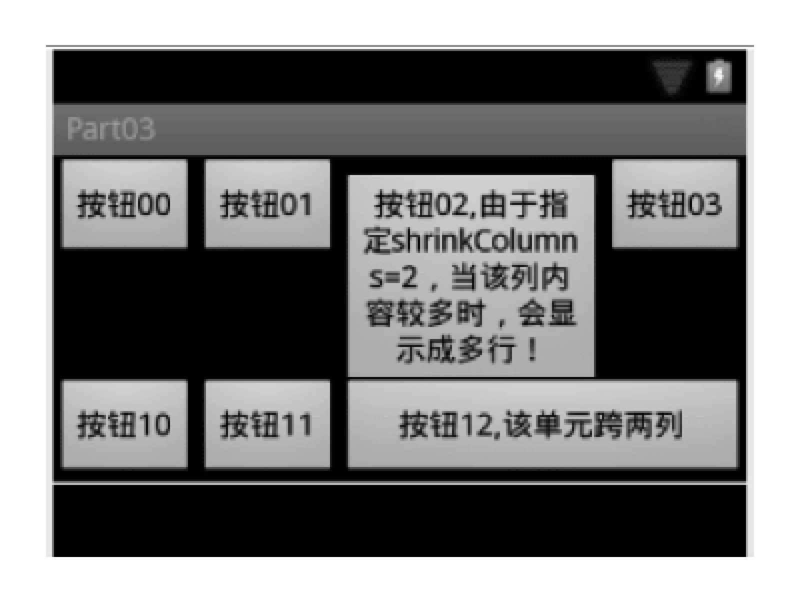


图 3.16　TableLayout 布局效果图

代码 03-13

```
<TableLayout
android:layout_width = "fill_parent"
android:layout_height = "wrap_content"
android:shrinkColumns = "2"
>
    <TableRow>
        <Button android:text = "按钮 00"/>
        <Button android:text = "按钮 01"/>
        <Button android:text = "按钮 02,由于指定 shrinkColumns = 2,当该列内容较多时,
                    会显示成多行!"/>
        <Button android:text = "按钮 03"/>
    </TableRow>
    <TableRow>
        <Button android:text = "按钮 10"/>
        <Button android:text = "按钮 11"/>
        <Button android:text = "按钮 12,该单元跨两列"
            android:layout_span = "2"
            />
    </TableRow>
    <ImageView
        android:background = "#ffffff"
        android:layout_height = "2px"
        />
</TableLayout>
```

上述代码中的表格是三行(两个 TableRow，一个 ImageView 独占一行)四列(由第 0 行确定，注意表格的行、列都从 0 开始)。属性 android:shrinkColumns="2"指明第 2 列具有收缩特性，其内容过多时会显示为多行，如果去掉该属性，第 2 列以后的内容会被挤压出屏幕可视区。属性 android:layout_span="2"指明该元素所在列占两列宽度。第三行是 ImageView 控件，占据一行的宽度。

TableLayout 不同于 Web 开发时的表格，没有边框线。在使用时注意以下几点。

- TableLayout 中的元素宽度由单元格决定，不能由控件的宽度决定。当指明宽度后，可以配合 android:stretchColumns="*"(所有列都具有伸展特性)达到平均分配各列的宽度，如图 3.17 所示。
- 在行中内容没有充满整行的情况下，被指定为具有伸展特性的列会自动伸展，以充满整行，否则没有效果。

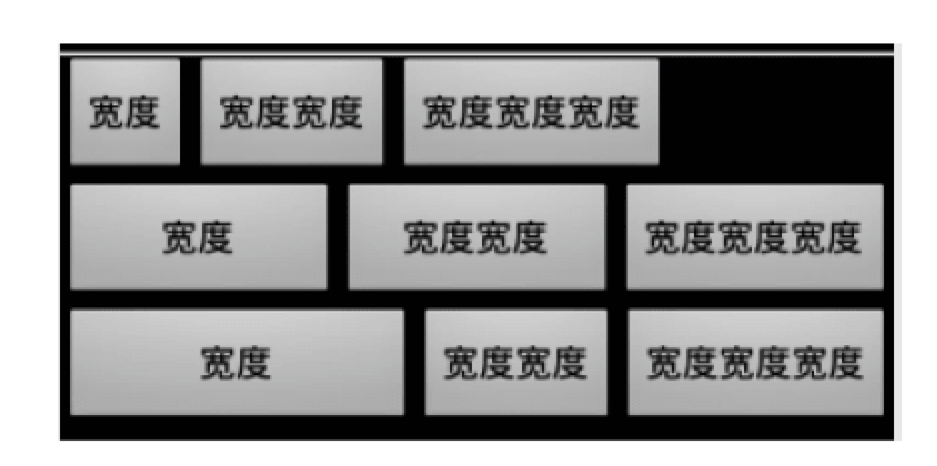


图 3.17　平分整行效果

3.4.4　FrameLayout

帧布局是 5 大布局中比较简单的一个，所有放入其中的元素都以左上角为参考，对齐父容器的左边和顶部，后放入的元素将覆盖前面的元素。帧布局常用于显示图片，充当应用程序背景，上面叠加应用程序。代码 03-14 演示 FrameLayout 的使用，完整代码参考 Part03 项目，testframelayout.xml 文件。

代码 03-14

```
<FrameLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent" >
    <ImageView
        android:src = "@drawable/pic4"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        />
    <TextView
        android:id = "@ + id/textView1"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "底层"
        android:textAppearance = "?android:attr/textAppearanceMedium" />
    <TextView
        android:id = "@ + id/textView2"
        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:text = "上层"
        android:textSize = "56sp"
        />
</FrameLayout>
```

3.4.5 AbsoluteLayout

绝对布局以屏幕左上角为坐标原点，水平向右为 x 轴正方向，竖直向下为 y 轴正方向。AbsoluteLayout 使用坐标点定位虽然精确，但代码灵活性较低，在分辨率不同的设备上会显示出不同的效果，因此这种布局一般不建议使用。AbsoluteLayout 常用的属性如下。

（1）android：layout_x，指定控件的 x 坐标。

（2）android：layout_y，指定控件的 y 坐标。

3.5 Image 和 Media

3.5.1 ImageView 与 BitmapFactory

ImageView 控件主要用于呈现图片，在很多场合都会用到。Android 支持的图片格式有 PNG、JPG 和 GIF 三种，将图片放入 res 文件下的 drawable-×××文件夹中，就可以通过“R. drawable 图片名”引用该图片。图片的命名要符合标识符命名规则，且不能使用大写字母。

ImageView 控件的常用属性如下。

（1）android:src，指定 ImageView 显示的图片，可以通过@drawable/图片名，引用项目中的资源。

（2）android:scaleType，图片的放缩模式。可以取以下值。

① matrix，采用用矩阵来绘图，从上到下、由左到右的顺序。

② fitXY，拉伸图片(不按比例)以填充 View 的宽高。

③ fitStart，把图片按比例扩大/缩小到视图的最小边，显示在视图的上部分位置。

④ fitCenter，按比例缩放图片到视图的最小边，居中显示。

⑤ fitEnd，按比例缩放图片到视图的最小边，显示在视图的下部分位置。

⑥ center，按原图大小显示图片，但图片宽高大于 View 的宽高时，中间部分显示。

⑦ centerCrop，按比例缩放图片，使得图片长(宽)大于等于视图的相应维度。

⑧ centerInside，当原图宽高小于或等于 View 的宽高时，按原图大小居中显示；反之将原图缩放至 View 的宽高居中显示。

给 ImageView 控件设置图片，可以直接使用 xml 属性，但大多数情况都是使用代码指定它的图片资源，这就要用到 BitmapFactory 类(工厂模式生产图片)。该类位于 android. graphics 包中，拥有众多静态方法，用于获取不同位置的图片。给 ImageView 的图片可以来源于项目、手机 SD 卡或是直接读取网络上图片。代码 03-15 演示直接给 ImageView 设定图片资源的方式，完整代码请参考 Part03 项目，MainActivity. java 源文件。

📖 **代码 03-15**

```
ImageView iv;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 ImageView
    setContentView(R.layout.layoutimageview);
    iv = (ImageView) findViewById(R.id.iv);
    //直接设定资源
    iv.setImageResource(R.drawable.pic4);
}
```

上述代码可以在程序运行时执行，设定图片资源，也可以通过代码 03-16 的方式设置图片资源，完整代码参考 Part03 项目，layoutimageview.xml 布局文件。这种方式设置的图片资源会被代码覆盖，所以如果该控件的图片不需要更换，可以直接在布局文件中指明。需要注意的是 android:src 属性和 android:background 属性都可以指定图片，但原理不同，前者是指定图片，位于上层，后者是指定背景，位于下层，且不支持缩放操作。

📖 **代码 03-16**

```
<ImageView
    android:id="@+id/iv"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:src="@drawable/pic17"
    android:background="@drawable/pic6"
    android:scaleType="centerInside"
    />
```

读取项目中的图片资源比较简单，但在应用上有很多的局限性。如果涉及图片资源过多，会造成打包时软件所占空间比较大。当图片比较多时，可以选择存放在 SD 卡中，需要使用时再读取即可。将其他资源创建成图片，需要使用 BitmapFactory 类，该类的方法都是静态的，返回值都是 Bitmap，常用的方法如下。

（1）decodeByteArray(byte[] data, int offset, int length)，从字节数字创建图片。

（2）decodeFile(String pathName)，从文件路径创建图片。

（3）decodeResource(Resources res, int id)，从 res 资源创建图片。

（4）decodeStream(InputStream is)，从输入流创建图片。

代码 03-17 演示使用 BitmapFactory 读取 SD 卡中的图片信息，并设置给 ImageView。完整代码请查看 Part03 项目，MainActivity.java 文件。

📖 **代码 03-17**

```
ImageView iv;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
        //测试 ImageView
        setContentView(R.layout.layoutimageview);
        iv = (ImageView) findViewById(R.id.iv);
        //直接设定资源
        //iv.setImageResource(R.drawable.pic4);
        //读取 SD 卡中图片
        FileInputStream fis = null;
        Bitmap img = null;
        if(Environment.getExternalStorageState()!= null){             //如果手机插卡
            File dir  = new File("/sdcard/pic");
            File pic = new File(dir,"pic6.jpg");
            try {
                if(dir.exists() &&pic.exists()){                       //资源存在
                    fis = new FileInputStream(pic);
                    Log.i("print", pic.getAbsolutePath());
                    img = BitmapFactory.decodeFile(pic.getAbsolutePath());
                    //img = BitmapFactory.decodeStream(fis);           //通过输入流构建 Bitmap
                    iv.setImageBitmap(img);
                }
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally{
                if(fis!= null)
                    try {
                        fis.close();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
            }
        }
    }
```

以上代码涉及读取 SD 卡的相关知识，具体内容在后续章节会有介绍。读取 SD 卡中图片，既可以通过图片路径，也可以通过输入流。代码 03-17 如果想运行并输出结果，需要在手机虚拟机的 SD 卡中先放入图片。

虚拟机默认不支持 SD 卡，在 Eclipse 菜单下的 Window 中选择 AVD Manager，选中虚拟机，单击 Edit 按钮，打开虚拟机编辑界面，在 HardWare 区域单击 New 按钮，添加新的硬件支持。选择 SD Card Support。这样虚拟机就可以模拟 SD 卡了。

选择菜单 Window→Open Perspective→DDMS 命令，切换到 DDMS 视图。打开 File Explorer 选项卡，如图 3.18 所示。展开 mnt 文件夹，sdcard 文件夹即对应虚拟机的 SD 卡根路径。通过右侧手机符号图标(push a file onto device)，就可以向虚拟机的 SD 卡中存入文件。

Threads | Heap | Allocation Tracker | File Explorer

Name	Size	Date	Time	Permissions	Info
data		2013-03-16	11:44	drwxrwx--x	
mnt		2013-04-25	08:03	drwxrwxr-x	
asec		2013-04-25	08:03	drwxr-xr-x	
obb		2013-04-25	08:03	drwxr-xr-x	
sdcard		2013-04-25	09:16	d---rwxr-x	
DCIM		2013-04-14	11:23	d---rwxr-x	
LOST.DIR		2013-03-16	08:34	d---rwxr-x	
baidu		2013-03-16	11:43	d---rwxr-x	
download		2013-03-16	11:39	d---rwxr-x	
pic		2013-04-25	09:17	d---rwxr-x	
secure		2013-04-25	08:03	drwx------	
system		2011-02-03	22:51	drwxr-xr-x	

图 3.18　虚拟机文件浏览界面

3.5.2　ImageButton

图片按钮直接继承 ImageView，具有按钮的功能，并可以根据需要自定义外观样式。ImageButton 的自定义外观主要有两种实现方式，一种是借助 XML 文件，一种是添加触屏事件。

1. 借助 XML 自定义样式

新建布局文件 layoutimagebutton.xml，如代码 03-18 所示，完整代码查看随书配套资料。ImageButton 的属性 android:src="@drawable/btstyle"指明控件采用的样式。此处 btstyle 是 drawable 文件夹中的一个文件，而不是图片。

代码 03-18

```
<ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#00000000"
        android:src="@drawable/btstyle"
        />
```

btsyle.xml 文件被当作可绘制资源使用。在 drawable 文件夹上单击右键，选择创建 Android XML 文件，此时创建的文件是 drawable 类型，并会自动放入 res 下 drawable 文件夹中。该文件自定义的代码如代码 03-19 所示，定义了按钮被单击、释放、获得焦点和一般情况下的样式。

📖 **代码 03-19**

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="false" android:drawable="@drawable/bt_release" />
    <item android:state_focused="true" android:drawable="@drawable/bt_release" />
    <item android:state_pressed="true" android:drawable="@drawable/bt_press" />
    <item android:drawable="@drawable/bt_release" />
</selector>
```

2. 使用触屏事件自定义样式

ImageButton 继承 View，获得了 OnTouchListener(触屏监听器)，实现监听器就可以完成自定义按钮样式。代码 03-20 演示如何使用 ImageButton 的触屏监听器，完整代码参看 Part03 项目，MainActivity.java 源文件。

📖 **代码 03-20**

```
ImageButton ibt;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 ImageButton
    setContentView(R.layout.layoutimagebutton);
    ibt = (ImageButton) findViewById(R.id.imageButton2);
    ibt.setOnTouchListener(new OnTouchListener(){
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            //Log.i("print", "toch");
            if(event.getAction() == MotionEvent.ACTION_DOWN){
                //触屏按下
                ibt.setImageDrawable(getResources().getDrawable(R.drawable.bt_press));
            }else if(event.getAction() == MotionEvent.ACTION_UP){
                //触屏释放
                ibt.setImageDrawable(getResources().getDrawable(R.drawable.bt_release));
            }
            return false;
        }
    });
}
```

触屏事件在后续章节中会有深入学习。该监听器中 onTouch 方法响应屏幕的操作，包括按下、抬起、滑屏等操作。所有事件都封装在 MotionEvent 类中。

3.6 Time 和 Date

3.6.1 TimePicker 和 DatePicker

TimePicker 和 DatePicker 都继承自 FrameLayout 类。时间选择控件向用户显示一天中的时间(可以为 24 小时，也可以为 AM/PM 制)，并允许用户进行选择。日期选择控件的

主要功能是向用户提供包含年、月、日的日期数据并允许用户对其修改。

TimePicker 控件通过 OnTimeChangedListener 监听器，处理时间改变事件；DatePicker 控件通过 OnDateChangedListener 监听器，处理日期改变事件。

代码 03-21 演示 TimePicker 与 DatePicker 的使用，完整代码参看 Part03 项目，MainActivity.java 文件，布局文件查看 layouttimedate.xml，运行效果如图 3.19 所示。



图 3.19 时间日期选择器

代码 03-21

```
TimePicker tp;
DatePicker dp;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 TimePicker 和 DatePicker
    setContentView(R.layout.layouttimedate);
    tp = (TimePicker) findViewById(R.id.timePicker1);
    dp = (DatePicker) findViewById(R.id.datePicker1);
    tv1 = (TextView) findViewById(R.id.td_t);
    tv2 = (TextView) findViewById(R.id.td_d);
   //
    tp.setIs24HourView(true);
    tp.setOnTimeChangedListener(new OnTimeChangedListener(){
        @Override
        public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
            tv1.setText("时间: " + hourOfDay + ":" + minute);
        }
    });
    //
```

```
        dp.init(2013, 8, 13, new OnDateChangedListener(){
            @Override
            public void onDateChanged(DatePicker view, int year,
                    int monthOfYear, int dayOfMonth) {
                tv2.setText("日期: " + year + " - " + (monthOfYear + 1) + " - " + dayOfMonth);
            }}
        );
    }
```

TimePicker 支持 24 小时和 AM、PM 时间格式，可以通过方法 tp.setIs24HourView(true)设置为 24 小时制式。DatePicker 改变事件的监听器需要通过 init 方法传入，并设置初始化的日期。需要注意的是 DatePicker 的月份是从 0 开始的。

3.6.2 Chronometer

Chronometer 是 TextView 的子类，可以用 1s 的时间间隔进行计时，并显示出计时结果。Chronometer 类有三个重要的方法：start、stop 和 setBase，其中 start 方法表示开始计时；stop 方法表示停止计时；setBase 方法表示重新计时。此外可以通过 setFormat 方法设置显示时间的格式。

代码 03-22 演示简要计数器的功能，完整代码请参看 Part03 项目，MainActivity.java 文件，布局文件为 layoutchronometer.xml，运行效果如图 3.20 所示。

代码 03-22

```
Chronometer chro;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //测试 Chronometer
    setContentView(R.layout.layoutchronometer);

    chro = (Chronometer) findViewById(R.id.chronometer1);
    chro.setFormat("计时: %s"); //%s 会被 Chronometer 的格式替换
    b1 = (Button) findViewById(R.id.c_start);
    b2 = (Button) findViewById(R.id.c_end);
    b3 = (Button) findViewById(R.id.c_reset);
    b1.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View v) {
            chro.start();
        }
    });
    b2.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View v) {
            chro.stop();
        }
    });
    b3.setOnClickListener(new OnClickListener(){
```

```
        @Override
        public void onClick(View v) {
            chro.setBase(SystemClock.elapsedRealtime());
        }
    });
}
```



图 3.20　计数器运行界面

3.6.3　AnalogClock 与 DigitalClock

Android 中的 AnalogClock 与 DigitalClock 都是时钟控件，前者是指针式时钟，后者是数字式时钟。这两个控件的使用比较简单，只需要在布局文件中添加控件，不需要编写代码。新建布局文件，如代码 03-23 所示，完整代码参看 Part03 项目，layoutclock.xml 文件，并将其设置给 MainActivity。

代码 03-23

```
<AnalogClock
    android:id="@+id/analogClock1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<DigitalClock
    android:id="@+id/digitalClock1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="DigitalClock" />
```

AnalogClock 控件继承 View，DigitalClock 继承 TextView，都是已经封装好的控件，只需要在指定位置放置控件即可。代码运行效果如图 3.21 所示。

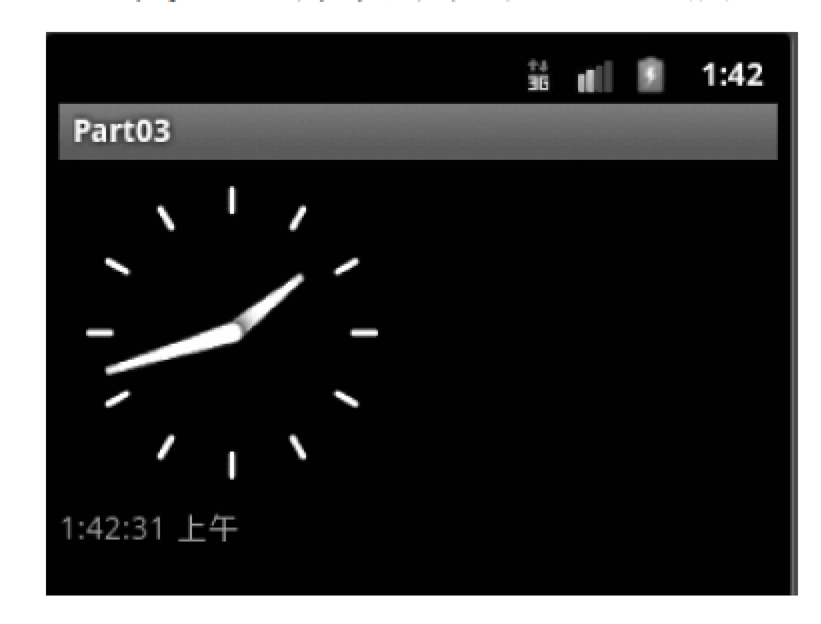


图 3.21　指针时钟与数字时钟

习　　题

1. Android 中控件的属性 layout_width 的取值有哪些？各有什么样的作用？

2. 编写一个 Hello World 程序，实现在屏幕上居中（垂直和水平两个方向）显示文本 Hello World，屏幕背景为黑色，字体颜色为白色。

3. 在日志中打印出当前日期和时间，在代码中产生一个异常（空指针、数组越界、除 0 等），捕获该异常并在日志中显示详细的异常信息。

4. 设计并实现数学计算机。

5. 请简要说明 android：stretchColumns 和 android：shrinkColumns 属性的作用。



CHAPTER 4

第4章

高级控件与数据适配器

本章主要内容:

ListView 控件;
ArrayAdapter、SimpleAdapter 与自定义适配器;
ExpandableListView;
GridView;
ScrollView 和 HorizontalScrollView;
SlidingDrawer;
TabHost;
Gallery 和 ImageSwitcher。

4.1 ListView 与适配器

ListView 是 Android 中比较常用的控件,它以列表的形式展示具体内容,并且能够根据数据的长度允许界面滚动。在使用 ListView 控件时,数据与布局样式是分离的,控件中的数据需要使用适配器 Adapter 添加,样式可以使用 Android 提供的系统样式,也可以自定义样式。

ListView 间接继承 android. widget. AdapterView 抽象类,获得了以下 4 种监听器。

(1) setOnClickListener(View. OnClickListener l),监听 ListView 控件的单击,一般不采用该监听器。

(2) setOnItemClickListener(AdapterView. OnItemClickListener listener),监听列表项单击操作。

(3) setOnItemLongClickListener(AdapterView. OnItemLongClickListener listener),监听列表项长时间单击。

(4) setOnItemSelectedListener(AdapterView. OnItemSelectedListener listener),监听列表项被选中操作。

AdapterView 继承了 android. widget. Adapter 接口。Adapter 接口的两个常用实现类

是 ArrayAdapter 和 SimpleAdapter，所以给 ListView 填充数据的任务就由 ArrayAdapter 或 SimpleAdapter 完成。如果需要自定义适配器，可以继承抽象类 BaseAdapter。

4.1.1 ArrayAdapter 适配器

ArrayAdapter 以 List 列表为数据源，可以采用 Android 系统风格完成简单样式的填充。使用适配器首先需要准备该适配器所需要的数据（ArrayAdapter 所需数据为 List），其次是创建 ArrayAdapter（此时需要指定样式），最后将适配器设定给 ListView。代码 04-1 演示 ListView 采用 ArrayAdapter 填充数据的过程，运行效果如图 4.1 所示。

代码 04-1

```
ListView lv;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //测试 EditText
    setContentView(R.layout.listviewlayout);
    //初始化 ListView
    lv = (ListView) findViewById(R.id.listView1);
    //Step1: 构造列表所需的数据
    List<String> data = new ArrayList<String>();
    data.add("曹操");
    data.add("司马懿");
    data.add("张飞");
    data.add("赵云");
    data.add("甘宁");
    data.add("孙尚香");
    //Step2: 构建适配器
    ArrayAdapter adapter = new ArrayAdapter(this,android.R.layout.simple_list_item_1,data);
    //Step3: 设置适配器
    lv.setAdapter(adapter);
}
```

ArrayAdapter 适配器有多个构造方法，详细信息可以查阅 API 文档。在构造适配器时需要提供的参数大体类似，需要 Context（上下文场景，一般指所在 Activity）、样式和数据。上述代码中构造 ArrayAdapter 时第一个参数即为当前上下文环境，第二个参数 android.R.layout.simple_list_item_1 是系统样式（之所以具有图 4.1 的样式，都是受此处影响），第三个参数是将要填充的数据，此时需要 List 类型的数据。

ArrayAdapter 一般只接受含有一个 TextView 的样式，这种类型的列表只能呈现由 List 提供的一行简单字符串。如果 List 中存放的不是字符串，而是对象类型，该对象类型可以通过重写 toString 方法，确定需要输出的字符串。如果一个 List 元素需要输出两个字符串信息，可以使用 android.R.layout.simple_list_item_2 样式，数据适配器应该采用 SimpleAdapter。



图 4.1　ArrayAdapter 填充 ListView

4.1.2　SimpleAdapter 适配器

SimpleAdapter 可以将数据映射到布局样式，其数据必须是 List 类型，且 List 的元素必须是 Map 类型，即满足 List<Map<string,? >>的泛型规则。SimpleAdapter 比 ArrayAdapter 功能更强，列表中的每一项对应一个 Map 项，Map 中不同 Key 对应的 Value 值可以映射到同一列表项的不同的控件视图上（这种列表样式也会更加复杂）。这些控件视图可以是 TextView、ImageView 或实现了 Checkable 接口（比如 CheckBox）。

通常情况下，TextView 控件需要对应的数据是字符串类型，即 Map 中 Value 值为字符串的 Key 可以绑定到该控件；ImageView 控件需要对应一个资源 id（图片资源的引用），即 Map 中 Value 为图片资源 id 引用的 Key 可以绑定到该控件；实现 Checkable 接口的控件需要对应 boolean 类型的数据。

修改代码 04-1，将需要填充的数据改为 Map 类型，如代码 04-2 所示，完整代码请参考 Part04 项目，MainActivity.java 文件。由于使用了 android.R.layout.simple_list_item_2 样式，列表的每一项由标题和内容两部分组成，对应两个 TextView 控件，其 ID 分别为 android.R.id.text1 和 android.R.id.text2。代码运行效果如图 4.2 所示。

代码 04-2

```
List<Map<String,String>> data = new ArrayList<Map<String,String>>();
Map map1 = new HashMap<String,String>();
map1.put("name", "曹操");
map1.put("type", "君主");
data.add(map1);
Map map2 = new HashMap<String,String>();
```

```
    map2.put("name", "司马懿");
    map2.put("type", "智将");
    data.add(map2);
    Map map3 = new HashMap<String,String>();
    map3.put("name", "张飞");
    map3.put("type", "武将");
    data.add(map3);
    Map map4 = new HashMap<String,String>();
    map4.put("name", "赵云");
    map4.put("type", "武将");
    data.add(map4);
    Map map5 = new HashMap<String,String>();
    map5.put("name", "甘宁");
    map5.put("type", "武将");
    data.add(map5);
    SimpleAdapter adapter = new SimpleAdapter(this,data,
            android.R.layout.simple_list_item_2,
            new String[]{"name","type"},
            new int[]{android.R.id.text1,android.R.id.text2});
```



图 4.2 SimpleAdapter 填充 ListView(一)

SimpleAdapter 适配器只有一个构造方法，需要传入 5 个参数。第一个参数是上下文环境，第二个参数是将要填充的数据，第三个参数是列表的样式(上述代码是采用系统样式，也可以自定义)，第四个参数和第五个参数是指明 Map(数据的元素类型都是 Map)中的数据如何与样式中的控件匹配，需要字符串数组和整型数组。字符串数组的值是 Map 中的 Key，整型数组是样式中控件的 ID，按照顺序将 Key 依次映射到控件上。

新建布局文件 listview_item_1.xml，作为列表的布局样式。修改代码 04-2 为代码 04-3，完整代码请查看 Part04 项目，MainActivity.java 文件。

📖代码 04-3

```
List<Map<String,String>> data = new ArrayList<Map<String,String>>();
Map map1 = new HashMap<String,String>();
map1.put("name", "曹操");
map1.put("type", "君主");
map1.put("info", "武力: 83 智力: 90");
map1.put("img", R.drawable.ccc);
data.add(map1);
 ⋮
SimpleAdapter adapter = new SimpleAdapter(this,data,
        R.layout.listview_item_1,                    //使用自定义样式文件
        new String[]{"name","type","info","img"},
        new int[]{R.id.listview_item_name,R.id.listview_item_type,
        R.id.listview_item_info,R.id.listview_item_img});
//设置适配器
lv.setAdapter(adapter);
```

上述代码中给每个元素新增加了两个 Key-Value，其中 img 对应的是图片资源的引用。在构造 SimpleAdapter 时，第三个参数不再使用系统样式，而是自定义的布局样式（参考布局文件 listview_item_1.xml），该布局文件采用水平线性布局，包含一个 ImageView 控件（ID 是 listview_item_img）和一个竖直线性布局，该竖直线性布局中又包含三个 TextView 控件（ID 分别是 listview_item_name、listview_item_type、listview_item_info）。Map 中的 4 个值（name、type、info、img）分别映射到以上 4 个控件中，映射顺序由第四个参数和第五参数决定，并按照布局文件中规定的样式排列，以达到如图 4.3 所示的效果。



图 4.3　SimpleAdapter 填充 ListView（二）

4.1.3　带有事件监听的 ListView

ListView 控件除了用于呈现数据之外，还可以通过事件监听与用户交互。常用的事件监听器是 OnItemClickListener、OnItemLongClickListener 和 OnItemSelectedListener，分

别处理列表项单击、列表项长按、列表项被选中事件。代码 04-4 演示了三种监听的使用，完整代码请查看 Part04 项目，MainActivity.java 文件。

ListView 控件添加了三个监听器，分别采用设置标题，弹出对话框，弹出 Toast 提示三种方式响应不同的事件。

OnItemClickListener 监听器响应用户的单击，执行 onItemClick 方法。该方法有 4 个参数，分别对应当前单击的 ListView；当前被单击的控件(ListView 的一项是对应相应控件)；被单击项是适配器的第几个元素；被单击的元素是 ListView 中的第几个。一般情况下，第三个参数和第四个参数是一样的。触发该事件的效果如图 4.4 所示。

OnItemLongClickListener 监听器与 OnItemClickListener 监听器类似，只是前者需要长按才会触发。OnItemLongClickListener 监听器的 onItemLongClick 方法也有 4 个参数，其含义与 OnItemClickListener 监听器中方法的参数类似。触发该事件的效果如图 4.5 所示。在该事件中使用了 Dialog 对话框，现在先不做深入介绍，对话框和通知等功能，在后续章节有介绍。



图 4.4 单击事件效果图



图 4.5 长按事件效果图

OnItemSelectedListener 监听器处理列表项被选中事件，这是事件在触屏手机上没有的效果，在带有按键的手机上，通过方向键改变选中元素时会触发。该监听器中采用 Toast 响应，后续章节中会有详细介绍。触发该事件的效果如图 4.6 所示。

代码 04-4

```
//给 ListView 添加监听器,使用匿名内部类实现监听
//添加列表项单击事件
lv.setOnItemClickListener(new OnItemClickListener(){
```



图 4.6　选中事件效果图

```
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                long arg3) {
            setTitle("第" + arg2 + "个元素被单击：" + data.get(arg2).get("name"));
        }
    });
    //添加列表项长按事件
    lv.setOnItemLongClickListener(new OnItemLongClickListener(){
        @Override
        public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {
            Builder builder = new Builder(MainActivity.this);
            builder.setTitle("角色人物选择：");
            builder.setIcon(android.R.drawable.ic_dialog_info);
            builder.setMessage("当前选中的是：" + data.get(arg2).get("name"));
            builder.show();
            return false;
        }}
    );
    //添加列表项选中事件
    lv.setOnItemSelectedListener(new OnItemSelectedListener(){
        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {
            Toast.makeText(MainActivity.this, data.get(arg2).get("name") + "被选中!",
                    Toast.LENGTH_LONG).show();
        }
```

```
    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
    }
});
```

4.1.4 自定义适配器

当 ListView 的列表项中出现按钮、复选框等附带事件的控件时，数据是无法映射到这些控件的。即使采用自定义布局文件显示这些控件，它们的事件监听器也不会响应，这就需要自定义数据适配器。

ArrayAdapter 和 SimpleAdapter 都是直接继承了 BaseAdapter 抽象类，如果要自定义适配器，也需要继承该类，并实现该类中的抽象方法。

(1) abstract int getCount()，返回适配器中的数据个数，确定列表有多少行。

(2) abstract Object getItem(int position)，获取指定位置的数据元素，一般不用。

(3) abstract long getItemId(int position)，获取指定位置元素的行号，一般不用。

(4) abstract View getView(int position, View convertView, ViewGroup parent)，绘制 ListView 中的每一项，这个方法比较重要，也比较复杂，自定义适配器体现在这里。

在 Part04 项目中新建类 SangoItemAdapter，继承 BaseAdapter，并重写上述 4 个抽象方法，如代码 04-5 所示，完整代码请查看 Part04 项目，SangoItemAdapter.java 文件。

代码 04-5

```
public class SangoItemAdapter extends BaseAdapter {
    private LayoutInflater inflate;                //用于将布局文件转化为视图
    //以下 5 个参数模仿 SimpleAdapter 的构造方法中的参数
    private Context context;
    private List<Map<String,Object>> list;         //用于存放数据
    private int res;                               //资源文件
    private String from[];
    private int[]to;
        public SangoItemAdapter(Context context,List<Map<String,Object>> list,
            int res,String from[],int to[]){
        this.context = context;
        this.list = list;
        this.res = res;
        this.from = from;
        this.to = to;
        inflate = LayoutInflater.from(context);
    }
    //需要
    @Override
    public int getCount() {
        return list.size();                        //用于确定 ListView 中需要多少个元素项
    }
```

```
    ⋮
    //需要,重要
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView img;
        TextView name, type, info;
        Button click;
        final int index = position;
        // 依次为每项数据所对应的控件
        convertView = inflate.inflate(res, null);
        img = (ImageView) convertView.findViewById(R.id.listview_item_img);
        img.setBackgroundResource((Integer)list.get(position).get("img"));
        name = (TextView) convertView.findViewById(R.id.listview_item_name);
        name.setText((String)list.get(position).get("name"));
        type = (TextView) convertView.findViewById(R.id.listview_item_type);
        type.setText((String)list.get(position).get("type"));
        info = (TextView) convertView.findViewById(R.id.listview_item_info);
        info.setText((String)list.get(position).get("info"));
        click = (Button) convertView.findViewById(R.id.listview_item_click);
        //给按钮 click 添加事件监听
        click.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                Builder builder = new Builder(context);
                builder.setTitle("删除人物");
                builder.setMessage("您是否确定要删除" + list.get(index).get("name"));
                builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        //执行删除操作
                    }
                });
                builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {

                    }
                });
                //显示对话框
                builder.show();
            }}
        );
        return convertView;
    }
}
```

根据以往学习的经验,但凡是自定义的东西,都要比系统提供可以直接使用的控件复杂。但自定义的组件更能够匹配特定项目、特定需求。自定义适配器中比较麻烦的是 getView 方法。第一个参数 position 标明需要绘制的行数,即数据的元素个数。第二个参

数为需要绘制的视图，即为 ListView 中每一项的布局。第三个参数一般不需要。

该方法中首先声明了列表每一项都需要的几个控件(ImageView、TextView、Button)；一个 final 修饰的 index，直接赋值 position(之所以需要这个 final 类型的变量，是因为在匿名内部类中只能引用方法中被 final 修饰的变量)。

convertView = inflate.inflate(res, null)，实现将列表项的布局映射成一个视图，这个视图中包含很多控件，通过使用 findViewById 方法，获取到这些控件的引用。然后将这些控件与相应列表项中的数据映射，通过 position 参数可以取得数据中的相应项。

最后给 Button 添加 OnClickListener，监听按钮的单击事件。在该监听器中构建了一个 Dialog 对话框(关于对话框在后续章节中介绍)，通过 index 变量标明该按钮对应的数据项。Dialog 中按钮比较特殊，这在深度讲解 Dialog 时再解析，目前就当照猫画虎即可。

修改 MainActivity.java 文件，使用自定义的适配器，如代码 04-6 所示，完整的代码请查看 Part04 项目，MainActivity.java 文件。自定义适配器的运行效果如图 4.7 所示。



图 4.7　自定义适配器的运行效果

代码 04-6

```
//使用自定义适配器
SangoItemAdapter sia = new SangoItemAdapter(this,data,
        R.layout.listview_item_2,
        new String[]{"name","type","info","img"},
        new int[]{R.id.listview_item_name,R.id.listview_item_type,
        R.id.listview_item_info,R.id.listview_item_img});
//设置适配器
//lv.setAdapter(adapter);
lv.setAdapter(sia);
```

4.2　ExpandableListView

扩展的列表控件，直接继承 ListView，功能要比 ListView 更加强大，不过在使用时比 ListView 稍微复杂。ExpandableListView 可以实现类似 QQ 折叠列表的效果，也有资料称为手风琴效果。稍加改动就可以实现较为实用的 UI。学习该控件的前提是熟练掌握自定义适配器的使用。

在 Part04 项目中新建 ExpandableActivity.java 文件，继承 ExpandableListActivity 类，自动得到 ExpandableListView 控件（由 ExpandableListActivity 类提供）。数据由 ExpandableListAdapter 适配器提供，该适配的常用子类是 SimpleExpandableListAdapter 和 BaseExpandableListAdapter。前者用于简单数据的适配，后者需要子类继承，自定义适配器。代码 04-7 演示了 ExpandableListView 控件使用自定义适配器填充数据，完整代码请查看 Part04 项目，ExpandableActivity.java 文件。

代码 04-7

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.expandablelayout);
    ExpandableListAdapter ela = prepareAdapter();
    setListAdapter(ela);
}
public ExpandableListAdapter prepareAdapter(){
    //构造存放组名的集合
    List<Map<String,String>> groups = new ArrayList<Map<String,String>> ();
    Map<String,String> group1 = new HashMap<String,String>();
    group1.put("groupName", "魏国");
    Map<String,String> group2 = new HashMap<String,String>();
    group2.put("groupName", "蜀国");
    Map<String,String> group3 = new HashMap<String,String>();
    group3.put("groupName", "吴国");
    groups.add(group1);
    groups.add(group2);
    groups.add(group3);
    //构造存放子元素的集合，这个泛型比较复杂
    List<List<Map<String,Object>>> childs  = new ArrayList<List<Map<String,Object>>>
();
    //构造第一组的数据
    List <Map<String,Object>> child1 = new ArrayList<Map<String,Object>>();
    Map<String,Object> item1 = new HashMap<String,Object>();
    item1.put("name", "曹操");
    item1.put("type", "君主");
    item1.put("info", "武力: 83 智力: 90");
    item1.put("img", R.drawable.ccc);
    child1.add(item1);
    Map<String,Object> item2 = new HashMap<String,Object>();
```

```
        item2.put("name", "司马懿");
        item2.put("type", "智将");
        item2.put("info", "武力: 56 智力: 95");
        item2.put("img", R.drawable.csmy);
        child1.add(item2);
        //构造第二组的数据,略
        …
        //构造第三组的数据,略
        …
        childs.add(child1);
        childs.add(child2);
        childs.add(child3)
        BaseExpandableListAdapter sela = new SangoExpandableAdapter(
            this,
            groups,
            R.layout.expandablegroup,
            new String[]{"groupName"},new int[]{R.id.groupName},
            childs,
            R.layout.listview_item_1,
            new String[]{"name","type","info","img"},
            new int[]{R.id.listview_item_name,R.id.listview_item_type,R.id.listview_item_
    info,R.id.listview_item_img}
            );
        return sela;
    }
```

ExpandableListView 控件分为两级,上级是列表组的显示,如 QQ 群组,展开组后是下级列表,如一个 QQ 组中的成员。ExpandableListAdapter 适配器既要提供组的数据、布局和匹配关系,又要提供成员列表的数据、布局和匹配关系,所以适配器的构造方法稍显复杂。

仔细阅读代码 04-7,适配器由方法 prepareAdapter 提供。该方法中主要构造了组的数据和成员的数据,都使用了泛型。最后新建自定义适配器 SangoExpandableAdapter,该适配器的详细代码请查看 Part04 项目,SangoExpandableAdapter.java 文件。自定义的适配器继承了 BaseExpandableListAdapter,并模仿 SimpleExpandableListAdapter 适配器,需要传入 9 个参数,含义如下。

(1) Context context,上下文运行环境,与 ListView 的适配中的含义一样。

(2) List<Map<String, String>> groups,组所需要的数据。

(3) int groupLayout,组布局样式,代码中此处采用 expandablegroup.xml 布局文件。

(4) String[] groupFrom,组数据中的 Key。

(5) int[] groupTo,组布局中控件 ID,与上一个参数匹配,实现数据到控件的映射。

(6) List<List<Map<String, Object>>> childData,成员列表数据。

(7) int childLayout,成员列表的样式,代码中此处采用 listview_item_1.xml 布局文件(ListView 曾用)。

(8) String[] childFrom,成员列表数据中的 Key。

(9) int[] childTo,成员列表样式中控件 ID,与上一个参数匹配,实现数据到控件的

映射。

相对于 ListView 的自定义适配器而言，该适配器多了两个层次的适配，既要匹配组的数据与样式，又要匹配组下列表成员的数据与样式。

由于 ExpandableListView 控件位于新建 ExpandableActivity 类中，项目运行前，需要将该类作为主 Activity 运行。修改 Manifest.xml 文件，将配置文件修改为如代码 04-8 所示，最终运行效果如图 4.8 所示。

代码 04-8

```
<application
    android:icon = "@drawable/ic_launcher"
    android:label = "@string/app_name" >
    <activity
        android:name = ".ExpandableActivity"
        android:label = "@string/app_name" >
        <intent - filter>
            <action android:name = "android.intent.action.MAIN" />
            <category android:name = "android.intent.category.LAUNCHER" />
        </intent - filter>
    </activity>
</application>
```



图 4.8 ExpandableListView 界面

4.3 GridView

与 ListView 不同，GridView 把元素按照二维表格的形式管理。一般用于显示图片、图标等，常见的九宫格界面使用 GridView 可以很容易实现。GridView 直接继承

AbsListView，是 View 的间接子类，常用的属性与作用见表 4.1。

表 4.1 GridView 常用属性与作用

属　性	作　用
android:columnWidth	用于设置列的宽度
android:gravity	列表项的对齐方法，与布局管理器中描述的一致
android:horizontalSpacing	两列之间的间距
android:verticalSpacing	两行之间的间距
android:numColumns	设置列数
android:stretchMode	缩放模式，取值 columnWidth、spacingWidth、spacingWidthUniform、null

GridView 使用 ListAdapter 填充数据，作为 ListAdapter 的子类 ArrayAdapter<T> 和 SimpleAdapter，都可以给 GridView 提供数据，如果需要自定义适配器，可以继承 BaseAdapter。GridView 对适配器的用法与 ListView 类似。

代码 04-9 演示使用 SimpleAdapter 给 GridView 提供数据，完整代码请查看 Part04 项目，GridViewActivity.java 文件，涉及的布局文件有两个：gridviewlayout.xml 和 gridview_item_1.xml。前者是作为 Activity 的布局文件，并提供 GridView 控件，后者是 GridView 数据项的布局样式。运行效果如图 4.9 所示。

代码 04-9

```
GridView gv;
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.gridviewlayout);
    gv = (GridView) findViewById(R.id.gridView1);
    //准备适配器
    SimpleAdapter sa = prepareData();
    //设置适配器
    gv.setAdapter(sa);
}
private SimpleAdapter prepareData() {
    List<Map<String,Object>> data  = new ArrayList<Map<String,Object>>();
    Map item1 = new HashMap();
    item1.put("img", R.drawable.ccc);
    item1.put("name", "曹操");
    …
    data.add(item1);
    …
    SimpleAdapter sa  = new SimpleAdapter(
            this,
            data,
            R.layout.gridview_item_1,
            new String[]{"img","name"},
```

```
            new int[]{R.id.gv_imageView1,R.id.gv_textView1}
    );
    return sa;
}
```



图 4.9　GridView 运行效果

gridview_item_1.xml 布局文件影响 GridView 数据项的样式，采用相对布局，TextView 控件位于 ImageView 控件的下方，都采用水平居中对齐。gridviewlayout.xml 布局文件定义了 GridView 控件的样式，属性 android:numColumns="3"指明行数为 3。对于布局复杂的 GridView 可以自定义适配器，请参考 4.1.4 节内容。

4.4　ScrollView 和 HorizontalScrollView

竖直滚动控件和水平滚动控件，是可供用户滚动的层次结构布局容器，可以通过滚动屏幕显示更多的内容。这两种滚动控件都直接继承 FrameLayout，这就意味控件中只能放置一个其他控件。对于复杂 UI 可以选择放置布局管理器，然后在布局管理器中再放置其他控件。

ScrollView 允许竖直滚动，通常用的子元素是垂直方向的 LinearLayout；HorizontalScrollView 允许水平滚动，通常用的子元素是水平方向的 LinearLayout。

4.5　SlidingDrawer

SlidingDrawer 可以通过拖动 handle 显示隐藏区域的内容，类似于抽屉的效果。该控件由两个 View 组成，一是可以拖动的 handle（以下称为抽屉的把手），一般用 Button 或 ImageButton 充当；其二是隐藏内容的 content（以下称为抽屉），一般使用布局管理器。使用 SlidingDrawer 必须在布局文件中指定 handle 和 content。

SlidingDrawer 常用的属性与作用见表 4.2，常用方法见表 4.3。

表 4.2 SlidingDrawer 常用属性与作用

属　　性	作　　用
android:allowSingleTap	是否可以通过 handle 打开或关闭，取值为 true 或 false
android:animateOnClick	指定当手柄打开或关闭 content 时是否该有一个动画
android:content	必需，指定内容 ID，供 content 控件引用
android:handle	必需，指定拖动按钮 ID，供 handle 控件引用
android:orientation	拖动按钮 ID 方向

表 4.3 SlidingDrawer 常用方法介绍

方　　法	作　　用
animateClose()	关闭抽屉时的动画
animateOpen()	打开抽屉时的动画
close()	立刻关闭抽屉
open()	立刻打开抽屉
getContent()	获取内容
getHandle()	获取控制
lock()	加锁，锁定后触屏事件将被忽略
unlock()	开锁，响应触屏事件

SlidingDrawer 控件直接继承 ViewGroup，间接继承 View，新增加三个内部监听器，功能如下。

(1) SlidingDrawer.OnDrawerCloseListener，监听抽屉关闭事件。

(2) SlidingDrawer.OnDrawerOpenListener，监听抽屉打开事件。

(3) SlidingDrawer.OnDrawerScrollListener，监听抽屉正在拖到事件。

代码 04-10 演示 SlidingDrawer 控件的使用，完整代码请查看 Part04 项目，SlidingDrawerActivity.java 文件，涉及的布局文件是 slidingdrawerlayout.xml 和 gridview_item_1.xml。

代码 04-10

```
SlidingDrawer sd;
ImageButton ib;
GridView gv;
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.slidingdrawerlayout);
    sd = (SlidingDrawer) findViewById(R.id.slidingDrawer1);
    ib = (ImageButton) findViewById(R.id.handle);
    gv = (GridView) findViewById(R.id.content);
    gv.setAdapter(prepareData());
```

```
        sd.setOnDrawerOpenListener(new OnDrawerOpenListener(){
            @Override
            public void onDrawerOpened() {
                setTitle("SlidingDrawer opened!");
            }}
        );
    }
```

slidingdrawerlayout.xml 布局文件的部分内容见代码 04-11。GridView 控件的使用参阅 4.3 节，prepareData()方法重用了 4.3 节中的方法。OnDrawerOpenListener 监听抽屉打开事件，当抽屉打开时将重设标题，根据需要可以添加其他三个监听器。

代码 04-11

```
<SlidingDrawer
    android:id = "@ + id/slidingDrawer1"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:content = "@ + id/content"
    android:handle = "@ + id/handle" >
    <ImageButton
        android:id = "@id/handle"
        android:layout_width = "60dip"
        android:layout_height = "40dip"
        android:src = "@drawable/folder_open"
        android:scaleType = "centerCrop"
        />
    <GridView
        android:id = "@id/content"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent"
        android:numColumns = "3"
        android:stretchMode = "columnWidth"
        android:verticalSpacing = "2dp"
        android:gravity = "center_horizontal"
        android:background = "@drawable/game_bg02"
        >
    </GridView>
</SlidingDrawer>
```

SlidingDrawer 控件中包含两个子控件：ImageButton 和 GridView。其属性 android：content＝"@＋id/content"和 android：handle＝"@＋id/handle"指明 handle 和 content 的 ID，分别由两个子控件引用。ImageButton 充当 handle 的角色，用于打开和关闭抽屉；GridView 就是所谓抽屉的布局。

作为 content 角色的控件一般是都是复合控件，如 GridView、ListView 或各种布局管理器，宽高常设为 match_parent，匹配打开的父容器。选择各个列表项的监听功能，与前面介绍的方法一样。代码的运行效果如图 4.10 和图 4.11 所示。

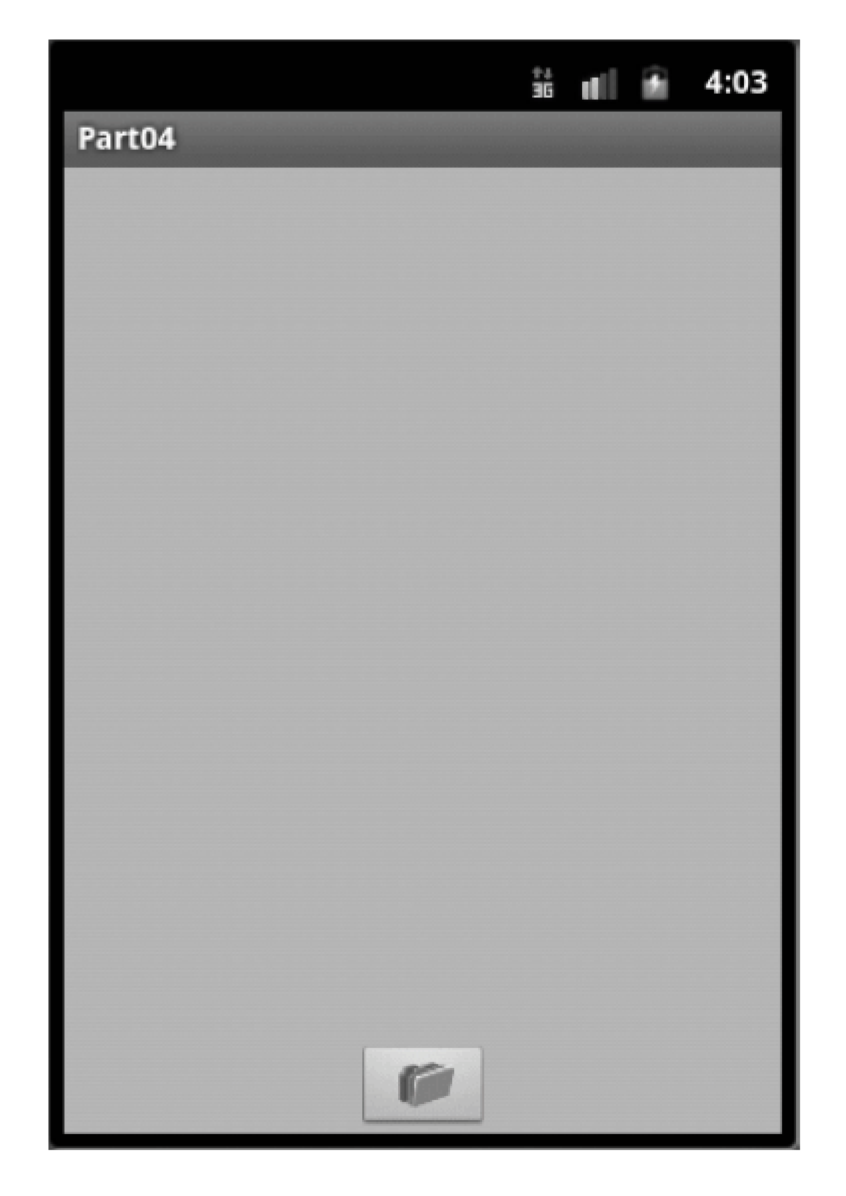



图 4.10 SlidingDrawer 关闭时界面



图 4.11 SlidingDrawer 打开时界面

4.6 TabHost 和 TabSpec

TabHost 是 Tab 的容器，包括两部分：TabWidget 和 FrameLayout。其中，TabWidget 就是每个 Tab 的标签，FrameLayout 则是 Tab 的内容。构建 TabHost 控件可以使用 Activity，指定布局文件，也可以通过继承 TabAcitivty，自动获得一个 TabHost 控件，此时如果再自定义样式布局，TabHost 的 android：id 必须设置为@android：id/tabhost，TabWidget 必须设置 android：id 为@android：id/tabs，FrameLayout 需要设置 android：id 为@android：id/tabcontent。

代码 04-12 演示继承 TabActivity 实现 TabHost 功能，完整代码请查看 Part04 项目，TabHostActivity. java 文件，布局文件是 tabhostlayout. xml。

代码 04-12

```
TabHost host;
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    host = getTabHost();                        //获取 TabHost 对象，由 TabActivity 提供
    LayoutInflater inflater = LayoutInflater.from(this);
    //将布局文件指定为 TabHost 的界面
    inflater.inflate(R.layout.tabhostlayout, host.getTabContentView());
    host.setBackgroundColor(Color.rgb(22, 77, 155));
    //下面创建 Tab
    TabSpec t1 = host.newTabSpec("t1");
```

```
        t1.setIndicator("未读信件",getResources().getDrawable(R.drawable.set_02));
        t1.setContent(R.id.tab_1);
        TabSpec t2 = host.newTabSpec("t2");
        t2.setIndicator("历史记录",getResources().getDrawable(R.drawable.set_01));
        t2.setContent(R.id.tab_2);
        TabSpec t3 = host.newTabSpec("t3");
        t3.setIndicator("系统设置",getResources().getDrawable(R.drawable.set_03));
        t3.setContent(R.id.tab_3);
        host.addTab(t1);
        host.addTab(t2);
        host.addTab(t3);
    }
```

继承 TabActivity 后通过 getTabHost()方法获取 TabHost 控件的实例，使用 LayoutInflater 将布局文件 tabhostlayout.xml 设定为 TabHost 的界面，其中的每个 Tab 可以采用该布局文件的相应控件或布局管理器。TabSpec 是 TabHost 控件中的选项卡，这是一个内部类，需要通过 host.newTabSpec()方法创建。在创建 TabSpec 时可以设定标题、标题的图标和内容面板。最后添加到 TabHost 中即可，添加的先后顺序决定了在 Activity 中的左右排列顺序，运行效果如图 4.12 所示。

在给 TabSpec 设定图标时，需要通过 getResources().getDrawable(R.drawable.set_03)，原因是 setIndicator 方法的第二个参数需要 Drawable 类型的，直接使用 R.drawable.set_03 资源是 int 类型的。另外，图标的大小最后小于 30×30，否则会出现标题与图标重叠。当然也可以通过代码 04-13 修改它们的位置。



图 4.12　TabHost 运行界面

📖 **代码 04-13**

```
TabWidget tw = tabHost.getTabWidget();
for (int i = 0; i < tw.getChildCount(); i++) {
    TextView tv = (TextView)tw.getChildAt(i).findViewById(android.R.id.title);
    ImageView iv = (ImageView)tw.getChildAt(i).findViewById(android.R.id.icon);
    iv.setPadding(0, -8, 0, 0);
    tv.setPadding(0, 0, 0, -2);
    tv.setTextSize(12);
}
```

4.7 Galley 和 ImageSwitcher

4.7.1 简单 Gallery

Gallery 控件常用于显示图片，在水平方向上支持图片的左右滑动，显示图片列表中的不同图片。当单击当前图像的后一个图像时，这个图像列表会向左移动一格，当单击当前图像的前一个图像时，这个图像列表会向右移动一格。也可以通过拖动的方式来向左和向右移动图像列表。Gallery 的数据也需要适配器提供，根据特定需求，可以继承 BaseAdapter，自定义适配器。

代码 04-14 演示了 Gallery 的简单用法，完整代码参考 Part04 项目，GalleryActivity.java 文件，布局文件是 gallerylayout.xml。（运行前请修改 Manifest，将 android:name 属性改为.GalleryActivity。）

📖 **代码 04-14**

```
public class GalleryActivity extends Activity {
Gallery gallery;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.gallerylayout);
    gallery = (Gallery) findViewById(R.id.gallery1);
    gallery.setAdapter(new GalleryImageAdapter(this));
    gallery.setOnItemClickListener(new OnItemClickListener(){
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                long arg3) {
            Toast.makeText(GalleryActivity.this, "当前选中第" + arg2 + "图片.",
                Toast.LENGTH_LONG).show();
        }}
    );
}
//内部类实现 自定义适配器
class GalleryImageAdapter extends BaseAdapter{
    Context context;
```

```
        //Gallery 用到的图片,使用项目中图片
        Integer pics[ ] = {R.drawable.ccc,R.drawable.csmy,R.drawable.lzf,R.drawable.lzy,
                R.drawable.ssq,R.drawable.sssx,R.drawable.sgn};
        public GalleryImageAdapter(Context c){
            context = c;
        }
        …
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            ImageView iv = new ImageView(context);
            iv.setImageResource(pics[position]);                //设置图片资源
            iv.setScaleType(ImageView.ScaleType.CENTER_CROP);   //放缩模式
            iv.setLayoutParams(new Gallery.LayoutParams(60,60)); //大小参数
            iv.setBackgroundColor(Color.LTGRAY);                //背景色
            return iv;
        }
        }
    }
```

Gallery 继承 AbsSpinner,本质上也属于列表性质的控件,实现水平方向展示列表项的功能。代码 04-14 中自定义适配器用于显示图片,这些图片都来自项目文件夹 res,也可以根据需要选择显示其他控件,或来自其他位置的图片。

getView 方法是自定义适配器中最为重要的方法,实现返回列表项中每一项的样式和数据。首先构建 ImageView 控件,随后设置该控件的属性,具体内容参考代码中的注释。代码运行效果如图 4.13 所示。



图 4.13　Gallery 运行效果

4.7.2 图片切换

单纯使用 Gallery 也可以实现图片切换，但是效果不是很好。ImageSwitcher 控件是 Android 中提供的专用于显示图片并支持切换效果的控件，用它实现图片的幻灯片播放可谓省时省力，而且效果很好。本节主要介绍使用 Gallery 与 ImageSwitcher 实现图片切换效果。

使用 ImageSwitcher 控件，必须设置一个 ViewFactory，用于将显示的图片和父窗口区分开，因此 Activity 需要实现 ViewSwitcher. ViewFactory 接口，通过 makeView()方法来显示图片，使用 setImageResource 用来指定图片资源。ImageSwitcher 控件还支持各种动画效果，可使用系统提供的动画，也可以自定义动画。关于动画会在后续章节介绍。

1. 设定布局

新建布局文件 imageswitcherlayout. xml，代码参考 04-15。设置外部为相对布局管理器，ImageSwitcher 填充父容器，并对齐顶部和左边。Gallery 位于父容器下边，设置高为固定 60dp，spacing 为 15dp 可以使得列表项的数据分开显示，避免重叠，背景带有透明效果。

代码 04-15

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageSwitcher
        android:id="@+id/imageSwitcher1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        >
    </ImageSwitcher>
    <Gallery
        android:id="@+id/is_gallery"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:gravity="center_vertical"
        android:spacing="15dp"
        android:background="#00cccccc"
        />
</RelativeLayout>
```

2. 新建 ImageSwitcherActivity. java

ImageSwitcherActivity 继承 Activity，实现 ViewFactory 接口和 OnItemClickListener

接口，参考代码 04-16，完整代码请查看 Part04 项目。

📖 **代码 04-16**

```
public class ImageSwitcherActivity extends Activity implements ViewFactory,OnItemClickListener {
    ImageSwitcher imageSwitcher;
    Gallery gallery;
    //由于两个控件都使用这一个图片组,就定义为属性
    Integer pics[ ] = {R.drawable.ccc,R.drawable.csmy,R.drawable.lzf,R.drawable.lzy,
     R.drawable.ssq,R.drawable.sssx,R.drawable.sgn};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.imageswitcherlayout);
        imageSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher1);
        imageSwitcher.setFactory(this);          //必需的,指定工厂,否则会有空指针
        gallery = (Gallery) findViewById(R.id.is_gallery);
        gallery.setAdapter(new GalleryImageAdapter(this));
        gallery.setOnItemClickListener(this);   //已实现监听器接口
        //ImageSwitcher 进入和退出时动画效果
         imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this, android.R.anim.
fade_in));
         imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this, android.R.anim.
fade_out));
    }
    //提供 ImageSwitcher 使用何种 View
    @Override
    public View makeView() {
        ImageView iv = new ImageView(this);
        iv.setBackgroundColor(0xffcccccc);
        iv.setScaleType(ImageView.ScaleType.FIT_CENTER);
        iv.setLayoutParams(new ImageSwitcher.LayoutParams(LayoutParams.MATCH_PARENT,
                    LayoutParams.MATCH_PARENT));
        return iv;
    }
    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        imageSwitcher.setImageResource(pics[arg2]);
    }
    //内部类实现 自定义适配器
    class GalleryImageAdapter extends BaseAdapter{
        …
    }
}
```

ViewFactory 接口只有一个方法 makeView，返回值为 View 类型，用于给 ImageSwitcher 提供视图。上述代码中在该方法创建 ImageView 对象，并设置相应属性，包括背景、放缩模式、尺寸。ImageSwitcher 使用 setFactory 方法设定此接口。注意，在使用 ImageSwitcher 前，一定要先设置 ViewFactory，否则会出现空指针异常。

Gallery 采用自定义适配器 GalleryImageAdapter，与前面的自定义适配器类似，具体代码请查看随书配套资料相应文件。

ImageSwitcher 控件设置了进入和退出动画，表现为图片切换时具有淡入和淡出的效果。Gallery 添加 OnItemClickListener 监听器，当 Gallery 相应的列表项被单击时，ImageSwitcher 切换到对应的图片，从而实现图片切换效果。代码运行效果如图 4.14 所示。

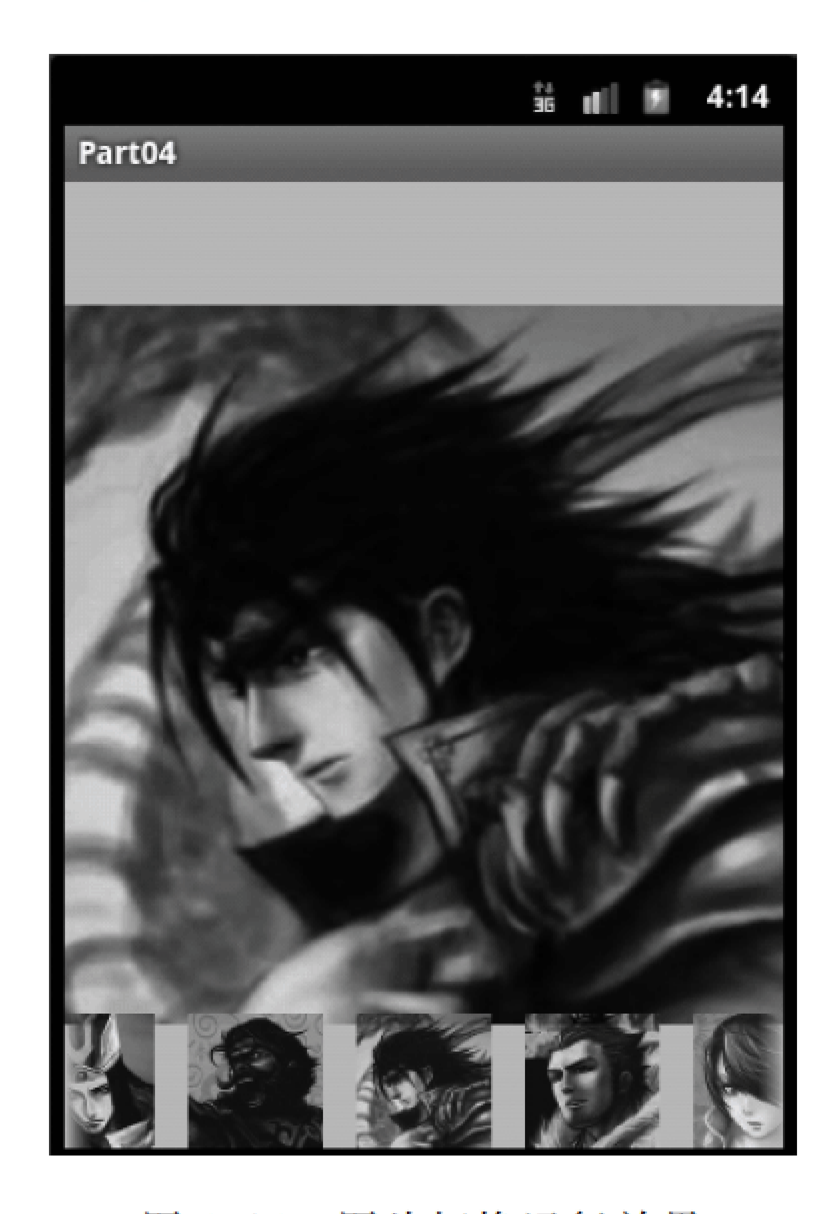


图 4.14 图片切换运行效果

习 题

1. 常用的数据适配器有哪些？它们都有什么样的特性？

2. 详细说明 SimpleAdapter 的构造方法 SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int resource, String[] from, int[] to)参数的含义是什么？

3. 简要说明下列参数是何种样式？

```
android.R.layout.simple_list_item_1
android.R.layout.simple_list_item_single_choice
android.R.layout.simple_list_item_multiple_choice
android.R.layout.simple_list_item_checked
```

4. 如何监听列表项的选择？

5．继承 BaseAdapter，实现如图 4.15 所示的界面布局。



图 4.15　习题 5 图



CHAPTER 5

第5章 使用系统组件

本章主要内容：

OptionMenu；
ContextMenu；
SubMenu；
AlertDialog 与 Builder；
自定义 Dialog；
Toast；
Notification；
ActionBar 的功能解析；
布局新方式 Fragment。

5.1 Menu

熟练使用 android.widget 包中的控件可以很好地实现 UI 布局，增强用户的交互体验。Android 除了提供前几章介绍的控件外，也支持丰富的菜单操作。在应用程序中灵活使用菜单提供帮助、导航、扩展其他操作或进行相应配置都是不错的选择。在不激活菜单的情况下，菜单不显示，可以将有限的屏幕空间让给最重要的控件，所以在移动设备应用程序开发中，菜单的使用非常重要。

Android 中常用的菜单有 OptionMenu、ContextMenu 和 SubMenu，都位于 android.view 包中，它们拥有不同的特性，下面对这些菜单逐一介绍。

5.1.1 OptionMenu

当用户单击移动设备上的菜单按钮(Menu)，屏幕底部将弹出一个菜单，触发事件弹出的菜单就是 OptionMenu(选项菜单)。选项菜单默认最多显示 6 个，这些菜单也被称作 IconMenu(带有图标的菜单)，当超过 6 个时，第 6 个菜单就会自动显示为“更多”/More，单击时可展开其他菜单(也称为 Expanded Menu)。

与 OptionMenu 密切相关的方法有以下几个。

(1) public boolean onCreateOptionsMenu(Menu menu)：创建 Menu 的方法，Activity 自带，重写该方法就可以实现创建菜单功能。

(2) public boolean onOptionsItemSelected(MenuItem item)：监听选中菜单项事件。

(3) public void onOptionsMenuClosed(Menu menu)：监听菜单关闭动作。

(4) public boolean onMenuOpened(int featureId, Menu menu)：监听菜单打开动作。

(5) public boolean onPrepareOptionsMenu(Menu menu)：菜单显示之前触发，可以进行菜单的调整。

代码 05-1 演示 OptionMenu 的创建，完整代码请查看随书配套资料 Part05 项目，MainActivity. java 代码文件，布局文件是 main. xml。

代码 05-1

```
public class MainActivity extends Activity {
    TextView tv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv_menu);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        tv.append("onCreateOptionMenu run.\n");
        menu.add(Menu.NONE, Menu.FIRST, Menu.FIRST, "新建")
                    .setIcon(android.R.drawable.ic_menu_add);
        menu.add(Menu.NONE, Menu.FIRST + 1, Menu.FIRST + 1, "保存")
                    .setIcon(android.R.drawable.ic_menu_save);
        menu.add(Menu.NONE, Menu.FIRST + 2, Menu.FIRST + 2, "删除")
                    .setIcon(android.R.drawable.ic_menu_delete);
        menu.add(Menu.NONE, Menu.FIRST + 3, Menu.FIRST + 3, "历史")
                    .setIcon(android.R.drawable.ic_menu_recent_history);
        menu.add(Menu.NONE, Menu.FIRST + 4, Menu.FIRST + 4, "信息")
                    .setIcon(android.R.drawable.ic_menu_info_details);
        menu.add(Menu.NONE, Menu.FIRST + 5, Menu.FIRST + 5, "帮助")
                    .setIcon(android.R.drawable.ic_menu_help);
        return true;
    }
    @Override
    public boolean onMenuOpened(int featureId, Menu menu) {
        tv.append("onMenuOpened run.\n");
        return super.onMenuOpened(featureId, menu);
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        tv.append("onOptionsItemSelected run." + item.getTitle() + "选中. \n");
        return super.onOptionsItemSelected(item);
```

```
        }
        @Override
        public void onOptionsMenuClosed(Menu menu) {
            tv.append("onOptionsMenuClosed run.\n");
            super.onOptionsMenuClosed(menu);
        }
        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
            tv.append("onPrepareOptionMenu run.\n");
            return super.onPrepareOptionsMenu(menu);
        }
    }
```

创建菜单项是通过 Menu 的 add 方法，该方法有 4 个重载方法，最为常用的是 add(int groupId, int itemId, int order, CharSequence title)，其他几个重载方法，读者可以翻阅 API 了解详情(理解了这个，其他几个都比较简单)。

方法第一个参数是菜单项的组号，如果不需要分组可以设为 null(参考代码，Menu. NULL 是 Menu 类中一个常量，表示不区分组)。当存在不同类别菜单项时，可以分组，按组分别处理。第二个参数是菜单项的 ID，应该具有唯一性，这个参数比较重要，是各菜单项区别的依据，当菜单项被单击时，可以通过它判断是哪个菜单。代码中使用 Menu. FIRST 常量，依次加 1 表示各菜单 ID，读者可以参考，也可以采用其他方案。第三个参数是菜单项在菜单上出现的先后顺序，序号最小的排在最前。第四个参数是菜单项的标题。

add 方法的返回值是 MenuItem，就是菜单项。MenuItem 提供较多的 set/get 方法，可以对菜单项进行相应的设置和获取属性。代码中使用 setIcon 设置菜单项的图标，这些图标都是 Android 提供的，特征就是以 Android. R 开头，也可以引用资源中的图片。代码运行效果如图 5.1 所示。



图 5.1　OptionMenu 运行效果

从运行的结果分析，onCreateOptionMenu 方法在程序运行中只会执行一次，用于创建菜单。菜单每次打开都会执行 onPrepareOptionMenu 和 OnMenuOpened 方法，所以对于菜单的调整可以放在这些方法中进行。当菜单每次关闭时都会执行 OnOptionsMenuClosed 方法。

5.1.2 SubMenu

Menu 的菜单项可以是 MenuItem，也可以是 SubMenu，MenuItem 直接显示在 Menu 上，SubMenu 也会直接显示在 Menu 上，但 SubMenu 可以包含二级菜单。SubMenu 主要用于将功能类似的一组菜单项组合在一起，进行多级显示。SubMenu 与 MenuItem 没有相关性，不能将 MenuItem 添加到 SubMenu。

SubMenu 继承 Menu，但与 Menu 有较大区别。SubMenu 可以设置图标，但它的下一级菜单不支持设置图标，只支持一个 HeaderIcon。SubMenu 的二级菜单类似于一个列表，HeaderIcon 就是列表头的概念。SubMenu 是通过 Menu 的 addSubMenu 方法添加的，该方法的返回值是 SubMenu，它使用从 Menu 继承到的 add 方法添加下一级菜单。

参考代码 05-2，修改代码 05-1 中的 onCreateOptionsMenu 方法，添加 SubMenu，并添加二级菜单。修改后运行效果如图 5.2 和图 5.3 所示。

代码 05-2

```
SubMenu sm = menu.addSubMenu(Menu.NONE, Menu.FIRST + 7, Menu.FIRST + 7, "查看");
sm.add(Menu.NONE, Menu.FIRST + 3, Menu.FIRST + 3, "历史");
sm.add(Menu.NONE, Menu.FIRST + 4, Menu.FIRST + 4, "信息");
sm.add(Menu.NONE, Menu.FIRST + 5, Menu.FIRST + 5, "帮助");
sm.setHeaderIcon(android.R.drawable.ic_menu_manage);
sm.setIcon(android.R.drawable.ic_menu_more);
```



图 5.2 SubMenu 运行效果(一)



图 5.3　SubMenu 运行效果(二)

关于 SubMenu 的使用请注意以下几点。

- SubMenu 与 MenuItem 是同一级别的菜单项,只是 SubMenu 可以包含子菜单。
- SubMenu 继承 Menu,得到了 addSubMenu 方法,但 SubMenu 不支持嵌套 SubMenu,会报异常信息 UnsupportedOperationException(不支持的操作)。
- SubMenu 的下一级菜单支持 onOptionsItemSelected 方法。

5.1.3　ContextMenu

ContextMenu 被称为上下文菜单,也叫作弹出式菜单。由于移动设备不同于 PC,可以单击右键,弹出菜单,所以 ContextMenu 一般由长按触发。ContextMenu 继承 Menu,但二者有较大区别,首先,ContentMenu 不支持设置图标,不能设置快捷键(对于触屏手机根本没有快捷键)。其次,Option Menu 的拥有者是 Activity,而上下文菜单的拥有者是 Activity 中的 View 控件。每个 Activity 有且只有一个 Option Menu,它为整个 Activity 服务。而一个 Activity 往往有多个 View,并不是每个 View 都有上下文菜单。

与创建 ContextMenu 密切相关的方法有以下几个。

(1) onCreateContextMenu(),在 Activity 中直接调用,使用 add 方法添加菜单项 MenuItem。

(2) onContextItemSelected(),响应菜单单击事件,这不同于 onOptionsItemSelected 方法。

(3) registerForContextMenu(),为 View 控件注册上下文菜单。

在代码 05-1 的基础上进行修改,重写方法 onCreateContextMenu 和 onContextItemSelected,参考代码 05-3,创建 ContextMenu,并添加响应操作。最后在 onCreate 方法中注册 ContextMenu,传入参数 tv,这就意味着将来运行时,在 tv 控件上长按,将弹出

ContextMenu，如图 5.4 所示。

📖 代码 05-3

```
TextView tv;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tv = (TextView) findViewById(R.id.tv_menu);
    registerForContextMenu(tv);             //为 TextView 控件注册 ContextMenu
}
//响应 ContextMenu 菜单项单击
@Override
public boolean onContextItemSelected(MenuItem item) {
    tv.setText(item.getTitle() + " 选择执行.");
    return super.onContextItemSelected(item);
}
//创建 ContextMenu 菜单
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderIcon(android.R.drawable.ic_menu_manage);
    menu.setHeaderTitle("请选择操作:");
    menu.add(Menu.NONE, Menu.FIRST, Menu.FIRST,"分享");
    menu.add(Menu.NONE, Menu.FIRST + 1, Menu.FIRST + 1,"删除");
    menu.add(Menu.NONE, Menu.FIRST + 2, Menu.FIRST + 2,"重命名");
}
```



图 5.4　ContextMenu 运行效果

onCreateContextMenu 创建菜单的过程与 onCreateOptionsMenu 方法类似，各参数的说明请参考 OptionMenu 的说明。View 参数是被注册控件，ContextMenuInfo 用于向 ContextMenu 传递额外信息，它依赖于 View 的类型。需要传递额外信息的 View 需要重写 getContextMenuInfo()方法，返回一个带有数据的 ContextMenuInfo 实现类对象。

5.2 Dialog

对话框在应用程序开发中使用比较频繁，可以增强界面的交互性，提升用户体验。Android 支持形式多样的对话框，既可以采用系统已提供的对话框样式，也可以自定义对话框样式。Dialog 类位于 android.app 包中，不过经常使用的是其子类 AlertDialog。

5.2.1 AlertDialog 与 Builder

AlertDialog 的构造方法被 protected 修饰，所以不能直接使用 new 关键字来创建 AlertDialog 类的对象实例。Builder 是 AlertDialog 的内部类，正如其名，用于建造 AlertDialog。一个通用的 AlertDialog 应该包括的内容有图标、标题、内容区域、按钮，这些组成部分都可以通过 Builder 构造。

Builder 类中常用的方法及作用简介，请参考表 5.1。

表 5.1 Builder 类中常用方法及作用

返回值	方法	作用
AlertDialog.Builder	setIcon	设置标题栏图标
AlertDialog.Builder	setTitle	设置标题
AlertDialog.Builder	setItems	设置内容区域为列表，以创建列表模式对话框
AlertDialog.Builder	setMessage	设置内容区域为文本信息，以创建信息模式对话框
AlertDialog.Builder	setMultiChoiceItems	设置内容区域为复选框，以创建多选模式对话框
AlertDialog.Builder	setSingleChoiceItems	设置内容区域为单选按钮，以创建单选模式对话框
AlertDialog.Builder	setView	设置内容区域为自定义布局，以创建自定义模式对话框
AlertDialog.Builder	setNegativeButton	左边按钮
AlertDialog.Builder	setNeutralButton	中间按钮
AlertDialog.Builder	setPositiveButton	右边按钮
AlertDialog	show	显示创建好的对话框

1. 简单对话框

在 Part05 项目中新建 DialogListActivity.java，继承 Activity，用来测试不同对话框的效果。AlertDialog 的创建参考代码 05-4，设置布局文件为 dialoglistlayout.xml，完整代码请参考随书配套资料 Part05 项目。

📖 **代码 05-4**

```
public class DialogListActivity extends Activity {
    Button b1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dialoglistlayout);
        b1 = (Button) findViewById(R.id.button1);
        //创建监听器并注册给控件
        BtClickListener listener = new BtClickListener();
        b1.setOnClickListener(listener);
    }
    //创建简单对话框
    public void createCommonDialog(View v){
        //创建 Builder
        AlertDialog.Builder builder  = new AlertDialog.Builder(this);
        //设置对话框的标题和图标
        builder.setIcon(android.R.drawable.ic_delete);
        builder.setTitle("删除信息");
        //设置信息
        builder.setMessage("请确认是否删除 " + v.getId());
        //左边按钮
        builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
            }
        });
        //右边按钮
        builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
            }
        });
        builder.show();
    }
    class BtClickListener implements OnClickListener{
        @Override
        public void onClick(View v) {
            if(v.getId() == R.id.button1){
                //创建简单对话框
                createCommonDialog(v);
            }
        }
    }
}
```

布局文件中只有一个 Button，当触发单击事件时，调用 createCommonDialog(v)，并将被单击的按钮传入，创建简单对话框。

对话框的标题和图标设定比较简单，直接调用相应方法即可。内容的设置可以根据相应的控件，视情况而定。最为复杂的是对话框按钮的添加，PositiveButton、NegativeButton和NeutralButton三个按钮可以不添加，也可以有选择地添加。添加按钮时的第一个参数是按钮的文本，第二个参数是按钮的监听器。对话框中按钮的监听器和一般按钮的监听器都是OnClickListener，但两个监听器不同。一般按钮的监听器是android.view.View.OnClickListener，对话框中按钮的监听器是android.content.DialogInterface.OnClickListener。Android中有很多诸如此类的用法，在写代码的过程中一定要注意是哪个包或哪个类中的监听器。

在代码05-4中为了区别两种监听器，创建对话框按钮监听器时，采用new DialogInterface.OnClickListener，直接指明监听器所处在的类。代码的最终运行效果如图5.5所示。



图5.5　简单对话框

2. 列表模式对话框

在dialoglistlayout.xml布局文件中添加第二个按钮，修改DialogListActivity.java，给新添加的按钮注册监听器，调用自定义创建对话框的方法，实现创建列表模式对话框，功能参考代码05-5，完整代码请参考Part05项目。

代码 05-5

```
//自定义创建列表模式对话框
    public void createListDialog(View v){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("请选择送货时间");
        builder.setIcon(android.R.drawable.ic_menu_directions);
        //设置列表，并注入监听器
```

```
            final String [ ]ss = {"星期一","星期二","星期三","星期四","星期五","星期六","星期天"};
            builder.setItems(ss, new DialogInterface.OnClickListener(){
                @Override
                public void onClick(DialogInterface dialog, int which) {
                     Toast.makeText(DialogListActivity.this,"您选择的时间是" + ss[which],
    Toast.LENGTH_SHORT)
                        .show();
                }}
            );
            //左边按钮
            builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    //
                }
            });
            //右边按钮
            builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    //
                }
            });
            builder.show();
        }
```

列表模式对话框没有信息区域,请不要使用 setMessage 设置文本信息,它使用一个列表代替文本信息。setItems 第一个参数是列表的数据项,需要传入字符串数组;第二个参数是监听器,此监听器也是位于 DialogInterface 中。代码运行效果如图 5.6 所示。



图 5.6 列表模式对话框

3. 复选模式对话框

在 dialoglistlayout. xml 布局文件中添加第三个按钮，id 为 Button3，修改 DialogListActivity. java，给新添加的按钮注册监听器，调用自定义创建对话框的方法，实现创建复选模式对话框，功能参考代码 05-6，完整代码请参考 Part05 项目。（注意，不要忘记控件的初始化和添加监听器。）

代码 05-6

```
//创建复选模式对话框
public void createMultipleDialog(View v){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("请选择曾用手机品牌");
    builder.setIcon(android.R.drawable.ic_menu_gallery);
    //设置列表,并注入监听器
    final String[]ss = {"联想","三星","苹果","诺基亚","黑莓","摩托罗拉","其他"};
    final List<String> item = new ArrayList<String>();
    builder.setMultiChoiceItems(ss, new boolean[]{false, false, false, false, false, false,
false},
              new DialogInterface.OnMultiChoiceClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which,
                boolean isChecked) {
            if(isChecked){
                item.add(ss[which]); //将选中的数据添加入列表
            } else{
                if(item.contains(ss[which]))item.remove(ss[which]);
            }
        }}
    );
    //左边按钮
    builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(DialogListActivity.this, "您选择的是" + item.toString(),Toast.
LENGTH_LONG)
                .show();
        }
    });
    //右边按钮
    builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            //
        }
    });
    builder.show();
}
```

复选模式对话框在列表对话框的基础上延伸了更强的功能，支持选择多项。setMultiChoiceItems 方法有三个参数需要设置，第一个是复选列表数据，需要一个字符串数组；第二个是 boolean 类型的数组，表示数据的选中状态，每一项都要给一个对应的 boolean 状态；第三个参数是监听器。

代码 05-6 中复选列表的监听器将选中的列表项，状态 isChecked 设为 true，添加到集合中。为实现选中该项后又取消的功能，还需要判断 isChecked 为 false 时，如果集合中存在该项，将该项移除。最终单击“确定”按钮后，显示所有选中的数据项。代码运行效果如图 5.7 所示。



图 5.7　复选模式对话框

4. 单选模式对话框

在 dialoglistlayout. xml 布局文件中添加第四个按钮，id 为 Button4，修改 DialogListActivity. java，给新添加的按钮注册监听器，调用自定义创建对话框的方法，实现创建单选模式对话框，功能参考代码 05-7，完整代码请参考 Part05 项目。（注意，不要忘记控件的初始化和添加监听器。）运行效果如图 5.8 所示。

代码 05-7

```
//创建单选模式对话框
public void createSingleDialog(View v){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("您的年龄");
    builder.setIcon(android.R.drawable.ic_secure);
    //设置列表，并注入监听器
    final String [ ]ss = {"总角之年","弱冠之年","而立之年","不惑之年","知命之年","耳顺之
年"};
```

```
    builder.setSingleChoiceItems(ss,2, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(DialogListActivity.this, "您选择的是 " + ss[which],Toast.LENGTH
_LONG)
            .show();
        }}
    );
    //左边按钮
    builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
        }
    });
    //右边按钮
    builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            //
        }
    });
    builder.show();
}
```



图 5.8　单选模式对话框

setSingleChoiceItems 方法用于设置单选，第一个参数是单选列表数据；第二个参数是默认选中哪一个数据项，数据项下标从 0 开始；第三个参数是监听器。

AlertDialog 的使用比较简单，但需要注意以下几点。

• 对话框中按钮监听器不同于一般按钮的监听器，在实现该监听器时不要引错类型。

- 对话框内容可以有多种形式，但不能同时呈现，否则无法显示。
- 对话框有三个按钮，如果添加了按钮，则单击任何按钮都可以退出对话框，如果没有添加按钮，需要按移动设备上的 Back 键，退出对话框。

5.2.2 ProgressDialog

进度条对话框是 AlertDialog 的子类，常用于比较耗时的操作，默认显示环形进度条。ProgressDialog 不需要通过 Builder 创建，可以直接使用构造方法创建。代码 05-8 演示环形进度条的使用，完整代码请参考 Part05 项目，DialogListActivity.java 文件。运行效果如图 5.9 所示。



图 5.9 环形进度条对话框

代码 05-8

```
//创建水平进度条对话框
public void createProgressDialog(){
    ProgressDialog pd = new ProgressDialog(this);
    pd.setIcon(android.R.drawable.ic_menu_upload);
    pd.setTitle("下载");
    pd.setMessage("文件下载中,请稍后…");
    pd.show();
}
```

上述代码可以更换为 ProgressDialog.show(this, "文件下载", "文件正在下载，请稍后……")，一句即可，运行效果与图 5.9 类似，只是图标默认采用系统信息图标。如果不设置标题，第二个参数可以赋值为空串。

ProgressDialog 默认是环形，可以通过方法 setProgressStyle(ProgressDialog.STYLE_HORIZONTAL)修改为水平进度条，如图 5.10 所示。水平进度条需要开启子线程，使用 Handler 修改主界面的 UI，可以参考第 4 章关于 ProgressBar 的讲解，也可以参考随书配套

资料 Part05 项目，DialogListActivity.java 文件，注意代码的注释与提醒。



图 5.10 水平进度条对话框

5.2.3 DatePickerDialog 和 TimePickerDialog

日期选择对话框和时间选择对话框，都继承 AlertDialog，不需要使用 Builder 创建，直接调用构造方法即可。DatePickerDialog 对话框和 TimePickerDialog 对话框的构造方法类似。都接受初始值，需要注入一个监听器，处理时间、日期改变事件。

（1）DatePickerDialog(Context context, DatePickerDialog.OnDateSetListener callBack, int year, int monthOfYear, int dayOfMonth)，其参数含义如下。

① context，上下文环境，对于 Activity 传入 this 即可。

② callBack，实现 OnDateSetListener 接口的监听器，处理选定日期事件。

③ year，初始年。

④ monthOfYear，初始月，注意 Android 返回的月是从 0 开始计算的。

⑤ dayOfMonth，初始日。

代码 05-9 演示 DatePickerDialog 的使用，使用 Calendar 赋初始日期值，当日期设定后，单击“设定”按钮，触发监听器，将 Activity 的标题设置为选定的日期，代码运行效果如图 5.11 所示。

代码 05-9

```
//创建日期选择对话框
public void createDatePickerDialog(){
    Calendar c = Calendar.getInstance();
    DatePickerDialog dpd = new DatePickerDialog(this,
            new OnDateSetListener(){
```

```
                    @Override
                    public void onDateSet(DatePicker view, int year,
                            int monthOfYear, int dayOfMonth) {
                        setTitle(year + "-" + (monthOfYear + 1) + "-" + dayOfMonth);
                    }},
                c.get(Calendar.YEAR),
                c.get(Calendar.MONTH),
                c.get(Calendar.DAY_OF_MONTH));
        dpd.setIcon(android.R.drawable.ic_dialog_alert);
        dpd.setTitle("请选择日期");
        dpd.show();
    }
```

（2）TimePickerDialog(Context context, TimePickerDialog.OnTimeSetListener callBack, int hourOfDay, int minute, boolean is24HourView)，其参数含义如下。

① context，上下文环境，对于 Activity 传入 this 即可。

② callBack，实现 OnTimeSetListener 接口的监听器，处理选定时间事件。

③ hourOfDay，设置小时。

④ minute，设置分钟。

⑤ is24HourView，是否以 24 小时制式显示。

代码 05-10 演示时间选择器的使用，完整代码请参考 DialogListActivity.java 文件，代码运行效果如图 5.12 所示。



图 5.11 日期选择对话框



图 5.12 时间选择对话框

📖 **代码 05-10**

```
//创建时间选择器
public void createTimePickerDialog(){
    TimePickerDialog tpd = new TimePickerDialog(
            this,
            new OnTimeSetListener(){
                @Override
                public void onTimeSet(TimePicker view, int hourOfDay,
                        int minute) {
                    setTitle("选定时间: " + hourOfDay + ":" + minute);
                }},
            18,16,true
    );
    tpd.setTitle("请选择时间");
    tpd.show();
}
```

5.2.4 自定义布局对话框

对话框除了上述布局，还支持使用 XML 布局文件自定义布局。自定义布局对话框比较重要，在很多情况下固有的对话框是无法满足要求的，比如实现登录对话框，这个时候自定义对话框的布局就可以派上用场。

自定义对话框的布局需要先定义布局文件，然后使用 LayoutInflater 将布局文件转换为视图 View，最后使用 Dialog 的 setView 方法将视图设为对话框的内容即可。请注意，这里介绍的只是自定义对话框的界面布局，如果需要自定义对话框的样式，比如字体大小、颜色、边框等，需要继承 Dialog 重写生成对话框的方法。

在 Part05 项目的 layout 文件夹中新建布局文件 logindialoglayout.xml，布局代码参考代码 05-11，竖直线性布局中嵌套两个水平线性布局，第一个存放登录名，由 TextView 和 EditText 组成；第二个存放登录密码，也是由 TextView 和 EditText 组成。自定义布局对话框的最终运行效果如图 5.13 所示。

📖 **代码 05-11**

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:padding = "10dp"
    android:layout_margin = "10dp"
    android:orientation = "vertical" >
    <LinearLayout
        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:orientation = "horizontal"
        >
        <TextView
            android:layout_width = "wrap_content"
```

```
                android:layout_height = "wrap_content"
                android:text = "账号:"
                />
            <EditText
                android:id = "@ + id/login_name"
                android:layout_width = "match_parent"
                android:layout_height = "wrap_content"
                android:singleLine = "true"
                android:hint = "输入账户"
                />
        </LinearLayout>
        <LinearLayout
            android:layout_width = "fill_parent"
            android:layout_height = "wrap_content"
            android:orientation = "horizontal"
            >
            <TextView
                android:layout_width = "wrap_content"
                android:layout_height = "wrap_content"
                android:text = "密码:"
                />
            <EditText
                android:id = "@ + id/login_password"
                android:layout_width = "match_parent"
                android:layout_height = "wrap_content"
                android:singleLine = "true"
                android:inputType = "textPassword"
                android:hint = "输入密码"
                />
            </LinearLayout>
</LinearLayout>
```



图 5.13　自定义布局对话框

修改 DialogListActivity. java 文件，增加第 8 个按钮，使其触发自定义对话框。注意初始化和添加监听器。自定义对话框布局的代码请参考代码 05-12。自定义布局对话框只有在设置对话框内容时不同，其他属性的设置不变。使用 LayoutInflater 可以将布局文件转换为视图，查找布局文件中的控件可以使用 findViewById。

代码 05-12

```
//创建自定义布局对话框
public void createLoginDialog(){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("用户登录");
    builder.setIcon(android.R.drawable.ic_menu_myplaces);
    //
    LayoutInflater inflater = LayoutInflater.from(this);
    View loginView = inflater.inflate(R.layout.logindialoglayout, null);
    final EditText name;
    final EditText pwd;
    name = (EditText) loginView.findViewById(R.id.login_name);
    pwd = (EditText) loginView.findViewById(R.id.login_password);
    builder.setView(loginView);
    builder.setPositiveButton("登录",new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which) {
            String n = name.getText().toString();
            String p = pwd.getText().toString();
            Toast.makeText(DialogListActivity.this, "登录信息：" + n + " " + p, Toast.LENGTH_
LONG)
                .show();
        }}
    );
    builder.setNegativeButton("取消",new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
        }
    });
    builder.show();
}
```

5.2.5 Dialog 样式的 Menu

对话框的显示需要依赖于 Activity，如果 Activity 为 Null，对话框会崩溃。为了更加简单地使用对话框，可以对其封装，以便于调用。对话框的使用比较灵活，在很多场合都可以用到。本节将介绍对话框式的菜单，让 OptionMenu 变成弹出式，也让对话框兼具菜单的作用。

新建 DialogMenuActivity. java 文件，作为程序运行的主界面。新建布局文件 gridviewdialog. xml，只含有一个 GridView 控件，作为对话框的界面。新建 gridviewdialog_item_1. xml，作为 GridView 控件的布局样式。布局文件的详细代码请查阅随书配套资料。

DialogMenuActivity. java 文件的核心功能参考代码 05-13。

📖 **代码 05-13**

```
public class DialogMenuActivity extends Activity {
    GridView gv;                          //对话框视图需要的控件
    AlertDialog dialog;                   //对话框
    List<Map<String,Object>> data;        //gv 需要的数据
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //初始化对话框,但不显示对话框
        initDialog();
    }
    //初始化对话框
    public void initDialog(){
        dialog = new AlertDialog.Builder(this).create();
         View dialogView = LayoutInflater.from(this).inflate(R.layout.gridviewdialog,
null);
        //设置对话框界面
        dialog.setView(dialogView);
        //获取对话框界面上的 GridView 控件
        gv = (GridView) dialogView.findViewById(R.id.dialog_gridview);
        gv.setAdapter(createMenuAdapter());
        gv.setOnItemClickListener(new OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                    long arg3) {
                Toast.makeText(DialogMenuActivity.this, "您选择的是第" + arg2 + "个菜单",
                                        Toast.LENGTH_LONG)
                    .show();
            }}
        );
        //注意此处不要调用 show
    }
    //创建 GridView 的数据适配器
    public SimpleAdapter createMenuAdapter(){
        data  = new ArrayList<Map<String,Object>>();
        int [] icons = {android.R.drawable.ic_menu_edit,android.R.drawable.ic_menu_delete,
                android.R.drawable.ic_menu_add,android.R.drawable.ic_menu_save,
                android.R.drawable.ic_menu_search,android.R.drawable.ic_menu_slideshow,
                android.R.drawable.ic_menu_share,android.R.drawable.ic_menu_manage};
        String []titles = {"编辑","删除","添加","保存",
                "查找","设为背景","分享","设置"};
        for(int i = 0;i < icons.length;i++){
            Map<String,Object> item = new HashMap<String,Object>();
            item.put("icon", icons[i]);
            item.put("title", titles[i]);
            data.add(item);
        }
        SimpleAdapter sa  = new SimpleAdapter(this,
                data,
                R.layout.gridviewdialog_item_1,
                new String[]{"icon","title"},
```

```
                new int[]{R.id.gridviewdialog_img,R.id.gridviewdialog_tv});
        return sa;
    }
    @Override
    public boolean onMenuOpened(int featureId, Menu menu) {
        dialog.show();
        return false;
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add("test");
        return super.onCreateOptionsMenu(menu);
    }
}
```

对话框的创建是在 Activity 创建时完成，但不直接显示，当按移动设备上的 Menu 键时，显示对话框，屏蔽 OptionMenu。这便是弹出式菜单运行的原理。

对话框的创建使用 new AlertDialog.Builder(this).create()，直接创建 AlertDialog，设置内容为一个 View 视图，该视图是通过 LayoutInflater 转换的布局文件 gridviewdialog.xml。查找对话框视图中的 GridView 控件，绑定数据适配器，适配器中的数据采用 Android 自带的系统图标，最后给 GridView 添加列表项监听器。

为了使得 Activity 依然响应设备上的 Menu 键，还需要重写 onCreateOptionsMenu 方法，该方法中的菜单已经没有什么意义，在显示前也会被屏蔽。不过为了让菜单完整，仍需要添加一个子菜单项。onMenuOpened 方法是菜单即将显示前调用的方法，该方法对上述程序比较关键，在此调用 dialog.show()显示对话框，使用 return false，屏蔽 OptionMenu，如果 return true，将显示 OptionMenu。代码运行效果如图 5.14 所示。注意，由于该对话框没有按钮，需要按设备上的 Back 键退出对话框。



图 5.14　对话框式的菜单

5.3 通 知

使用对话框可以实现向用户通知信息的目的，但对话框会打断当前操作，让用户转到对话框界面。如果需要的只是提醒，或通知用户某项工作已经开始，Android 还提供了另外两种向用户通知的方式：Toast 和 Notification。相对而言，Toast 的使用非常简单，前面已经反复使用过 Toast；Notification 稍微复杂，本节将介绍以上两种通知的使用。

5.3.1 Toast

Toast 主要用于向屏幕上弹出一条提示信息，不影响用户的其他操作，不会打断用户与控件的交互。Toast 不接受其他事件，在屏幕停留指定时间后，自动消失。Toast 可以从 Activity 或 Service（第 6 章介绍）创建并显示，在 Activity 上创建的 Toast，直接显示在该 Activity 上；在 Service 中创建的 Toast，会显示在当前活动的 Activity 上。

1. 简单 Toast

创建 Toast 比较简单，可以通过构造方法，也可以使用 Toast 的静态方法 makeText（以上章节中使用 Toast 都是采用这种方式）。Toast 的常用方法及作用详见表 5.2。

表 5.2 Toast 常用方法及作用

方 法 名	作 用
Toast(Context context)	唯一的构造方法，可以用于创建 Toast 对象，必须使用 setView 设置视图
makeText	最为常用的静态方法，返回值是 Toast，用于直接创建 Toast 对象
setDuration	Toast 在屏幕上停留的时间
setGravity	Toast 在屏幕上的位置
setText	Toast 显示的文本信息
show	显示 Toast

代码 05-14 演示使用构造方法创建 Toast，在使用 show 显示 Toast 之前，必须为 Toast 设置 View，否则抛出运行时异常。该部分的测试代码对应配套资料中的 Part05 项目，ToastActivity.java 文件，布局是 toastlayout.xml。

代码 05-14

```
Toast toast = new Toast(ToastActivity.this);
toast.setDuration(3000);
TextView tv = new TextView(ToastActivity.this);
tv.setText("构造 Toast,显示 3 秒");
toast.setView(tv);
toast.show();
```

显示简单的 Toast，可以使用 Toast 的静态方法 makeText（Context context, CharSequence text, int duration）。第一个参数是上下文环境，通常是 Activity 或 Application；第二个参数是要显示的文本信息；第三个参数是 Toast 在屏幕上的停留时间，

可以设为数值,单位毫秒,也可以使用 Toast 的静态属性 LENGTH_LONG 或 LENGTH_SHORT。代码 05-14 可以替换为如下代码,效果一致。

```
Toast.makeText(ToastActivity.this, "静态方法", Toast.LENGTH_SHORT).show();
```

2. 指定显示位置

Toast 模式显示的位置是屏幕下半部居中,可以通过 setGravity 方法修改显示位置。setGravity 方法必须在 show 方法之前调用。setGravity(int gravity, int xOffset, int yOffset),第一个参数是位置信息,如 Gravity.TOP(顶部)或 Gravity.CENTER(居中),具体可以查阅 Gravity 静态属性;第二个参数是相对于第一个参数指定的位置,在 X 轴方向的偏移量,取正值则向右偏移,反之左移;第三个参数是 Y 轴方向的偏移,效果类似第二个参数。

3. 定制显示界面

Toast 默认只能显示一个文本信息,如果需要显示多样性的界面,需要重新设置布局,指定布局文件,使用 LayoutInflater 转换为 View 视图,然后设置给 Toast 即可。代码 05-15 演示如何自定义 Toast 的布局,完整代码可以查阅 Part05 项目,ToastActivity.java 文件。

代码 05-15

```
View toastView = LayoutInflater.from(ToastActivity.this)
    .inflate(R.layout.toastinterfacelayout, null);
//设置布局中的控件内容
TextView tv = (TextView) toastView.findViewById(R.id.toast_msg);
tv.setText("自定义 Toast 信息");
ImageView iv = (ImageView) toastView.findViewById(R.id.toast_img);
iv.setImageResource(R.drawable.android01);
Toast toast = new Toast(ToastActivity.this);
toast.setView(toastView);                    //设置 Toast 视图
toast.setDuration(Toast.LENGTH_LONG);
toast.show();
```

使用 LayoutInflater 将布局文件转换为视图,还可以实现更为复杂的 Toast 布局,如设置标题等。但对于常规的 Toast 多用于显示文本信息,使用最简单的方式实现就可以了,复杂信息的提示完全可以由 Dialog 实现,简单提示这才是使用 Toast 的真谛。上述代码的运行效果如图 5.15 所示。

5.3.2 Notification

对于前台运行的 Activity 可以通过对话框、Toast 向用户发出提示信息,而后台运行的程序,如下载、收到信息等应用,则需要使用 Notification(通知)向用户发出提示信息。

Notification 发出的信息,包括图标、标题、时间可以显示在手机的状态条上。展开状态条,显示系统状态栏,详细的文本内容会显示在手机的状态栏中,单击通知可以通过预设的



图 5.15 自定义 Toast 运行效果

Intent 跳转到相应的应用程序。除此之外，Notification 还可以发出声音、灯光等信号。

Activity 和 Service 都可以发出 Notification，但对于 Activity 来说，它执行事件时都处于前台运行状态，不需要发出通知，所以创建并发出通知，一般都是在 Service 中进行的（Service 在第 6 章中介绍，此处依然用 Activity 发出通知）。当用户正在使用其他应用程序或设备处于休眠状态，就可以通过发出通知，提示用户处理相应事件。创建并发出通知需要用到的类有 Notification、Notification.Builder 和 NotificationManager。

Notification 可以通过调用构造方法创建，也可以使用 NotificationBuilder（类似 AlertDialog）创建。Notification 支持设置图标、标题、时间等信息。NotificationManager 是系统服务，用于管理 Notification。NotificationManager 无法直接创建，可以通过 getSystemService 获取实例。

实现向状态栏发送通知，需要获取 NotificationManager 对象，创建 Notification，设置 Notification 的相关属性，准备一个 Intent，最后发出该通知，具体过程可以遵循以下 4 步。

（1）获取 NotificationManager 对象。

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);
```

（2）创建 Notification 对象，可以使用构造方法，Android 3.0 以后，建议使用 Builder 创建。

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();
Notification notification = new Notification(icon, tickerText, when);
```

(3) 设置信息,准备 PendingIntent,这些信息用于展开状态栏后显示。

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);
```

(4) 将通知发给 NotificationManager。

```
private static final int HELLO_ID = 1;
mNotificationManager.notify(HELLO_ID, notification);
```

代码 05-16 演示如何发送一个简单的 Notification,展开状态栏,单击通知后,会跳转到 ToastActivity 界面。完整代码请查看 Part05 项目,NotificationActivity.java 文件。注意,该应用程序需要跳转到其他 Activity,这些 Activity 需要在 Manifest 文件中配置。

代码 05-16

```
public class NotificationActivity extends Activity {
    Button b1;
    NotificationManager nmanager;
    Notification notification;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notificationlayout);
        b1 = (Button) findViewById(R.id.notification_bt1);
        b1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                sendNotification();
            }}
        );
    }
    //发送通知
    public void sendNotification(){
        //1.获得 NotificationManager
        nmanager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        //2.创建 Notification
        notification = new Notification(
            R.drawable.folder_open,
            "收到文件",
            System.currentTimeMillis()
            );
        //3.设置属性,这些属性会在展开状态栏后显示
        Intent intent = new Intent(this,ToastActivity.class); //转向其他 Activity
        PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent,0);
```

```
            notification.setLatestEventInfo(this, "接收文件", "文件已经下载完成", pIntent);
            //4.将 Notification 发给 Manager
            nmanager.notify(1, notification);
        }
    }
```

Notification 的构造方法 Notification(int icon, CharSequence tickerText, long when),第一个参数是显示在状态条上的图标;第二个参数是显示在状态条上的文本;第三个参数是时间。运行的效果如图 5.16 所示。

setLatestEventInfo(Context context, CharSequence contentTitle, CharSequence contentText, PendingIntent contentIntent)设置通知在状态栏展开后的效果,第一个参数是 Context;第二个参数是通知的标题;第三个参数是通知的内容;第四个参数是当用户单击该通知后跳转的目的地。运行效果如图 5.17 所示。



图 5.16 收到通知

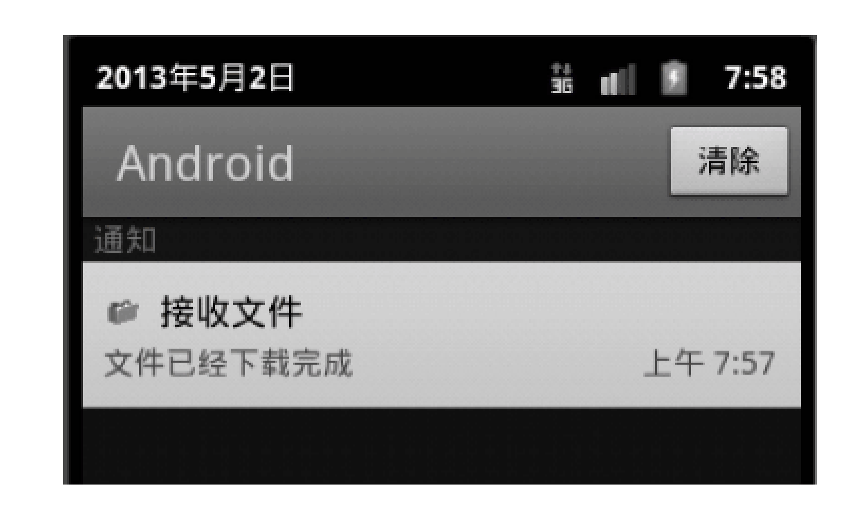


图 5.17 状态栏中的通知

发送通知由 NotificationManager 完成,使用 notify(int id, Notification notification)方法,第一个参数是通知的 ID。如果要发送很多通知,ID 应该是不同的,相同 ID 会执行更新操作,前提是上一个通知没有被清除。比如连续收到同一个号码的两个信息,在第一个还没有被处理之前,可以通过更新通知的方式提示用户,这要比重新发送一个通知更加方便。notify 方法的第二个参数是待发送的通知。

Notification 除了可以发出文本、图标提示之外,还可以增加播放声音、震动、亮灯信号,更加方便地通知用户。下面介绍如何增加这些内容。请注意,在测试时需要使用真机,虚拟机无法测试这些功能。

（1）带有声音的通知。

Notification 可以使用系统默认的通知声音，也可以指定播放音频。声音默认播放一次，也可以设置循环播放，直到用户处理通知。使用默认声音的代码为：

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

指定播放声音文件的代码为：

```
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
```

设置声音循环播放，直到用户处理通知的代码为：

```
notification.flags |= Notification.FLAG_INSISTENT;
```

（2）带有震动的通知。

Notification 通知产生时，增加震动效果，可以选择默认或是指定震动方式。震动不能循环。采用默认震动方式的代码为：

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

自定义震动方式的代码为：

```
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

数组第一个元素是震动前等待时长，第二个元素是震动时长，第三个元素是停止时长，第四个元素是再次震动时长。

（3）带有闪灯的通知。

闪灯可以默认，也可以自定义，不同的设备会有不同的响应方式。增加默认闪灯的代码为：

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
```

5.3.3 定制 Notification

默认的通知包含一个 ImageView 控件，两个 TextView 控件。使用 RemoteViews 可以自定义 Notification 的布局，这些布局会呈现在状态栏中。RemoteViews 可以使用指定的布局文件，转换为视图，然后向该视图中添加布局中的控件，最后将 RemoteViews 设定给 Notification，其他操作与默认 Notification 类似。

代码 05-17 演示自定义 Notification 的创建过程，详细代码可以查阅 Part05 项目，NotificationActivity.java 文件，指定的自定义布局文件是 notificationinterfacelayout.xml，代码运行效果如图 5.18 所示。



图 5.18 自定义 Notification

📖 **代码 05-17**

```
//发送自定义通知
    public void sendCustomNotification(){
        //1.获得 NotificationManager
        nmanager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        //2.创建 Notification
        notification  = new Notification(
            R.drawable.folder_open,
            "收到文件",
            System.currentTimeMillis()
            );
        RemoteViews rv = new RemoteViews(getPackageName(),R.layout.notificationinterfacelayout);
        rv.setImageViewResource(R.id.notification_img, R.drawable.savefile);
        rv.setTextViewText(R.id.notification_title, "催眠曲.mp3");
        rv.setProgressBar(R.id.notification_progressbar, 100, 10, false);
        notification.contentView = rv;
        //3.设置属性,这些属性会在展开状态栏后显示
        Intent intent  = new Intent(this,ToastActivity.class); //转向其他
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP|Intent.FLAG_ACTIVITY_NEW_TASK);
        PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, 0);
        notification.contentIntent = pIntent;
        //4.将 Notification 发给 Manager
        nmanager.notify(notificationID++, notification);
    }
```

RemoteViews 的构造方法需要传入两个参数：报名和自定义布局文件。该类提供了很多 set××× 方法，可以实现添加多种控件的功能，代码中添加了 ImageView 和 ProgressBar。设定完 RemoteViews 之后，将该对象赋值给 Notification 的 contentView 属

性。自定义的 Notification，不能使用 setLatestEventInfo 方法。PendingIntent 对象也需要直接赋值给 Notification 的属性 contentIntent，这两点不同于默认 Notification 的使用。最后使用 NotificationManager 的 notify 方法发送通知。

5.4 ActionBar

Android 操作系统在 3.0 之后引入两个比较重要的更新，一个是 ActionBar，另一个是 Fragment。随着智能手机的发展，很多手机厂商出品的手机已经不再具备 Menu 按键，这使得前面介绍的菜单操作无法实现。Android 操作系统不在强制要求手机具备 Menu 按键，而选择使用 ActionBar 完成菜单功能，当然 ActionBar 要比前面介绍的菜单功能更加完善。ActionBar 的主要作用体现在以下几个方面。

（1）完成显示菜单项的功能。

（2）使得应用程序的 ICON 变成向上回退的导航或直接返回手机桌面的按键。

（3）直接添加交互式控件。

（4）提供标签导航，方便不同的 Fragment 之间切换。

（5）提供下拉列表导航功能。

5.4.1 启用 ActionBar

ActionBar 完全可以取代 OptionsMenu 的功能，但是在 Android 中如果要使用 ActionBar 组件，必须使用 Android 3.0 及以后的版本。当配置文件中 android:minSdkVersion 或者 android:targetSdkVersion 属性被设置成 11 或者更高时，该应用会被认为是 Android 3.0 版本及以上，此时的 Activity 中默认含有 ActionBar 组件，否则在程序中获取 ActionBar 时，返回 null。

如果在某个 Activity 中需要关闭 ActionBar，可以对该 Activity 的配置信息设置主题样式为＜activity android:theme="@android:style/Theme.Holo.NoActionBar"＞；也可以通过以下三个方法控制 ActionBar 的显示与隐藏。

（1）show()，如果 ActionBar 没有显示，则显示 ActionBar。

（2）hide()，如果 ActionBar 处于显示状态，则隐藏 ActionBar。

（3）isShowing ()，如果 ActionBar 处于显示状态，则返回 true。

ActionBar 位于 android.app 包中，是一个抽象类，不能直接创建。当 Android API 版本在 v11 及以上时，可以通过 Activity 的 getActionBar()方法，直接获取 ActionBar 对象。



图 5.19 ActionBar 的不同区域

当 ActionBar 启用之后，应用程序就不存在 TitleBar（标题栏）。实际上 ActionBar 起到了标题栏和菜单融合的作用，并添加了导航功能。图 5.19 显示了一个 ActionBar 的 4 个区域，每个区域都能实现不同的功能。第 1 个区域是应用程序图标（ICON，请留意图标的左边有箭头），ActionBar 可以实现将该图标用作向上导航（返回前一个 Activity），或直接回到手机桌面的功能。第 2 个区域是应用程序的标题，这与原来的标题栏一致。第 3 个区域可以添加显示的菜单（实际上就是

OptionMenu 的菜单项 MenuItem)，当菜单比较多时，可以隐藏到第四个区域，相当于 OptionMenu 中的 More 菜单项。

5.4.2 处理 Action 菜单

Android 操作系统在 3.0 之后已经不再强制要求手机具有 Menu 按键，对应的原来的菜单可以由 ActionBar 完成。将一个普通的菜单项添加到 ActionBar，作为 ActionItem 使用，有两种方式。一种是在定义 Menu 文件时，使用 android:showAsAction 属性，该属性可以指定 5 种取值，如果需要取两个值，中间使用|连接，注意不能有空格。

(1) never，该菜单项不显示在 ActionBar 上。

(2) ifRoom，当 ActionBar 上有足够的空间时，显示该菜单项。

(3) always，一直显示该菜单项。

(4) withText，菜单项的图标和文本信息都显示在 ActionBar 上。(菜单项默认只显示图标。)

(5) collapseActionView，将 ActionView 折叠为普通的菜单项。

另一种给 ActionBar 添加菜单项的方法是直接使用代码。API 11 之后 MenuItem 类可以使用方法 setShowAsAction (int actionEnum)来确定如何在 ActionBar 上显示。参数 actionEnum 的取值可以是以下几个，它们的意义与上述内容一致。

(1) SHOW_AS_ACTION_ALWAYS。

(2) SHOW_AS_ACTION_IF_ROOM。

(3) SHOW_AS_ACTION_NEVER，这是默认项。

(4) SHOW_AS_ACTION_WITH_TEXT。

新建项目 Part05_1，选择 Android 3.0 或以上的版本。在 res/menu 文件夹中，新建 Menu 的布局文件 main.xml，如代码 05-18 所示。

代码 05-18

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- 定义一个可单选的菜单项 -->
    <item
        android:title="医疗服务"
        android:icon="@drawable/medical05"
        android:showAsAction="always|withText"
         >
         <menu>
             <group android:checkableBehavior="single">
                 <item
                     android:id="@+id/menu_item1"
                     android:icon="@drawable/medical01"
                     android:title="病例记录"
                      />
                 <item
                     android:id="@+id/menu_item2"
```

```
                    android:icon = "@drawable/medical02"
                    android:title = "查询药物"
                     />
                <item
                    android:id = "@ + id/menu_item3"
                    android:icon = "@drawable/medical04"
                    android:title = "预约检测"
                     />
            </group>
        </menu>
    </item>
    <item
        android:id = "@ + id/menu_item4"
        android:icon = "@drawable/medical03"
        android:title = "紧急呼救"
        android:showAsAction = "ifRoom|withText"
        />
</menu>
```

在 MainAtivity.java 中重写方法 onCreateOptionsMenu()，用于创建菜单，并重写 onOptionsItemSelected()方法，响应不同菜单单击。onCreateOptionsMenu 方法的代码如代码 05-19 所示。

代码 05-19

```
//通过菜单文件创建菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

项目的运行效果如图 5.20 和图 5.21 所示。当横屏后 ActionBar 有足够的空间，所以将 MenuItem 的文本信息也显示出来了。在该 Menu 的布局定义中，只设置了两个菜单项：



图 5.20 竖屏 ActionBar 效果

医疗服务和紧急呼救。其中,医疗服务菜单是一个单项列表按钮,单击时会展开单选下拉列表。本次运行并未出现图5.19所描述的第4个区域,原因是Menu布局中定义的菜单比较少,没有隐藏其他菜单的必要。但菜单项增加到ActionBar无法容纳时,第4个区域将自动出现。

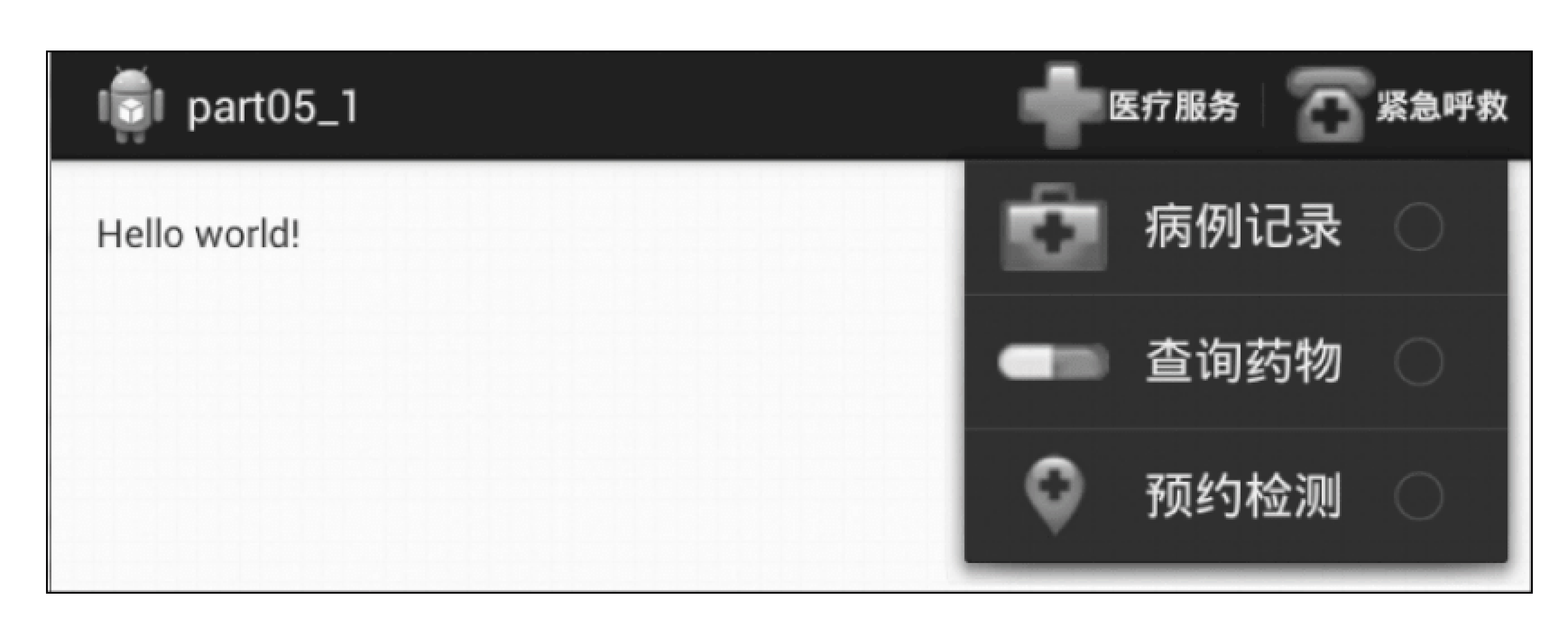


图5.21 横屏ActionBar效果

5.4.3 启用应用程序图标

ActionBar的第一个区域是应用程序的图标,默认情况下,该图标不可以单击,也就无法处理响应功能。在ActionBar类中有如下方法可以启用应用程序图标,当应用程序图标启用之后,就可以当作一般的菜单项来处理了。应用图标的ID默认是android.R.id.home。

```
actionBar.setDisplayHomeAsUpEnabled(true);
```

setDisplayHomeAsUpEnabled(boolean showHomeAsUp),启用应用程序图标,使其可以向上返回,此时应用程序图标的左边会出现箭头。在项目Part05_1中增加MedicalActivity.java,当单击“病例记录”菜单时,跳转到该Activity,然后使用图片返回,代码如代码05-20所示。

代码 05-20

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_medical);
    actionBar = getActionBar();
    //启用应用程序图标
    actionBar.setDisplayHomeAsUpEnabled(true);
    Log.i("Tag", "MedicalActivity onCreate 执行");
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()){
    case android.R.id.home:                         //应用程序图标被单击
```

```
            Intent intent = new Intent(MedicalActivity.this,MainActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            break;
        }
        return true;
    }
```

MedicalActivity 的运行效果如图 5.22 所示,通过应用图标可以返回上一个 Activity。

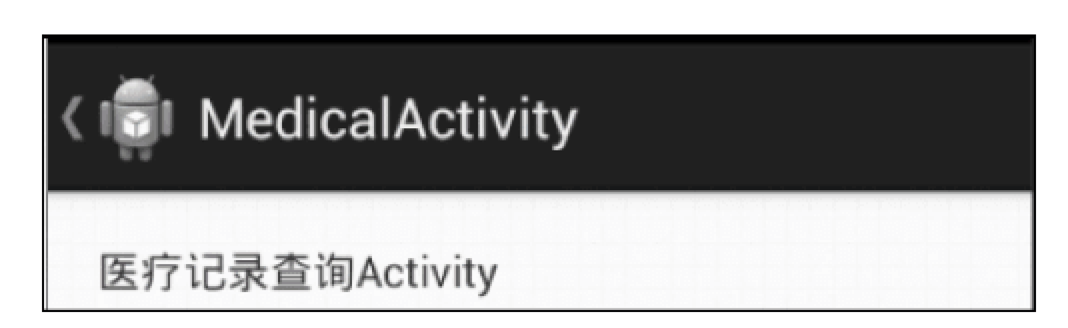


图 5.22 启用应用程序图标的效果

5.4.4 添加可交互视图

ActionBar 的强大之处还体现在它允许添加可交互的视图控件。在 ActionBar 上添加交互控件有两种方式,代码 05-21 是第一种方式,通过 actionViewClass 属性,直接指定所需添加控件的类;第二种方式是采用属性 android:actionLayout="@layout/searchview",给 actionLayout 属性赋值布局文件,该文件可以自定义。

代码 05-21

```
<! -- 添加一个交互控件,将会出现在 ActionBar 上 -- >
< item
    android:id = "@ + id/menu_find"
    android:title = "查询"
    android:showAsAction = "always"
    android:actionViewClass = "android.widget.SearchView"
    />
```

在 ActionBar 上添加完交互控件后,可以采用代码 05-22 的方式获取该控件对象。ActionBar 上的交互控件 ActionView 的运行效果如图 5.23 和图 5.24 所示。

代码 05-22

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //从菜单文件创建菜单
    getMenuInflater().inflate(R.menu.main, menu);
    //初始化 SearchView
    sv = (SearchView) menu.findItem(R.id.menu_find).getActionView();
    sv.setIconifiedByDefault(true);
    return true;
}
```

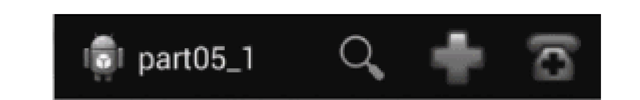


图 5.23 ActionView 折叠状态

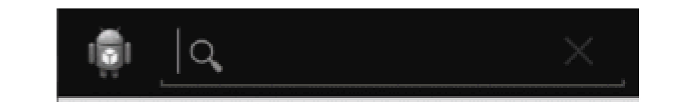


图 5.24 ActionView 展开状态

5.4.5 标签导航

上述 ActionBar 都是针对标题栏的改造，除此之外，ActionBar 还可以实现标签导航。ActionBar 可以使用 Tab 在不同的 fragment（关于 Fragment 的内容在 5.5 节介绍）之间切换。此处所介绍 Tab 与 4.6 节介绍的 TahSpec 稍有不同。

使用 ActionBar 的 Tab 导航功能需要设置 setNavigationMode(int mode)，导航模式，其中参数 mode 的取值为：

（1）NAVIGATION_MODE_STANDARD，标准导航，本节前面所采用的都是标准导航模式及菜单模式。

（2）NAVIGATION_MODE_LIST，下拉列表导航。

（3）NAVIGATION_MODE_TABS，标签导航。

如果使用标签导航，则必须将 mode 设为 NAVIGATION_MODE_TABS，然后调用 ActionBar 的方法 addTab(ActionBar.Tab tab)添加标签，该方法有多个重载，都比较简单，在此就不再列出。ActionBar.Tab 为抽象类，不能直接创建，所以参数 tab 需要通过调用 ActionBar.Tab newTab ()方法获取 Tab 对象。Tab 类的 set×××()方法也都会返回一个 Tab 对象，如 ActionBar.Tab setText(CharSequence text)。

新建项目 part05_2，完整项目请查阅随书配套资料，MainActivity.java 如代码 05-23 所示。代码中所用 FragmentA、FragmentB 和 FragmentC 都是 Fragment 的简单用法，在 5.5 节将详细介绍 Fragment 类。ActionBar 的标签导航和 Fragment 的联合使用，能够很好地完成显示界面的切换。

代码 05-23

```
public class MainActivity extends Activity {
    ActionBar actionBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mainlayout);
        //v11 之后，可以直接获取 ActionBar 对象
        actionBar = getActionBar();
        //确定 Tab 导航
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        //创建选项卡
        ActionBar.Tab t1 = actionBar.newTab();
        t1.setText("时政要闻");
        ActionBar.Tab t2 = actionBar.newTab();
        t2.setText("科技信息");
        ActionBar.Tab t3 = actionBar.newTab();
        t3.setText("体坛资讯");
```

```
        //给选项卡添加监听器
        MyTabListener mtl = new MyTabListener();
        t1.setTabListener(mtl);
        t2.setTabListener(mtl);
        t3.setTabListener(mtl);
        //将选项卡添加到 ActionBar
        actionBar.addTab(t1);
        actionBar.addTab(t2);
        actionBar.addTab(t3);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    //自定义 Tab 监听器
    class MyTabListener implements ActionBar.TabListener{
        @Override
        public void onTabReselected(Tab tab, FragmentTransaction ft) {

        }
        //当某个 Tab 被选中时,执行该方法
        @Override
        public void onTabSelected(Tab tab, FragmentTransaction ft) {
            Log.i("Tag", tab.getPosition() + " " + tab.getText());
            Fragment fragment = null;
            switch(tab.getPosition()){
            case 0:
                fragment = new FragmentA();
                break;
            case 1:
                fragment = new FragmentB();
                break;
            case 2:
                fragment = new FragmentC();
                break;
            }
            ft.replace(R.id.fragment_place, fragment);  //根据不同标签,更换 Fragment
        }
        @Override
        public void onTabUnselected(Tab tab, FragmentTransaction ft) {
        }
    }
}
```

项目运行效果如图 5.25 和图 5.26 所示。当竖屏时,ActionBar 空间受限,会自动将 Tab 标签移到下一行;当横屏时,ActionBar 空间足够,则 Tab 标签会位于 ActionBar 上。



图 5.25 竖屏 ActionBar 标签

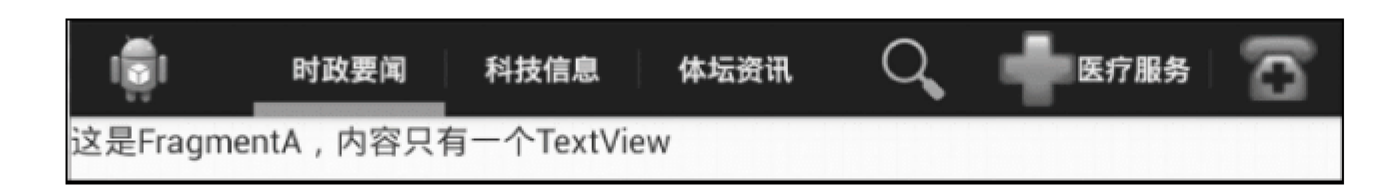


图 5.26 横屏 ActionBar 标签

5.4.6 下拉导航

在 ActionBar 上实现下拉式导航的步骤与 Tab 导航类似，首先需要设置 setNavigationMode(int mode)，导航模式为 NAVIGATION_MODE_LIST。然后调用 ActionBar 的 setListNavigationCallbacks()方法，设置下拉列表的数据适配器和下拉选项选中时的监听器，如代码 05-24 所示。

代码 05-24

```
//确定列表导航
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
//新建列表适配器
ArrayAdapter aa = new ArrayAdapter(this,R.layout.listlayout,
        R.id.listlayout_text,new String[]{"时政要闻","科技资讯","体坛快报"});
//设置列表导航的数据和监听器
actionBar.setListNavigationCallbacks(aa, new MyNavigationListener());
```

代码 05-25

```
//自定义 OnNavigationListener
class MyNavigationListener implements ActionBar.OnNavigationListener{
    @Override
    public boolean onNavigationItemSelected(int itemPosition, long itemId) {
        Log.i("Tag", itemPosition + " " + itemId);
        return false;
    }
}
```

下拉列表的监听器如代码 05-25 所示。onNavigationItemSelected()方法中的参数 itemPosition 是数据适配器中数组的下标，根据不同的选项，执行响应的操作即可。运行效果如图 5.27 所示。



图 5.27 ActionBar 下拉列表导航

5.5 Fragment

Fragment 是 Android 3.0 之后引入的系统组件，主要目的是在大屏幕设备(Android 3.0 之后支持平板)上支持更加动态和灵活的 UI 设计。较大的屏幕有更多的空间来放更多的 UI 组件，并且这些组件之间会产生更多的交互。

使用 Fragment 设计类似于新闻浏览的界面将变成非常简单的事情。一个新闻应用可以在屏幕左侧使用一个 Fragment 来显示一个文章的列表，在屏幕右侧使用另一个 Fragment 来显示详细内容。而这在以前需要分在两个 Activity 中实现，如图 5.28 所示。

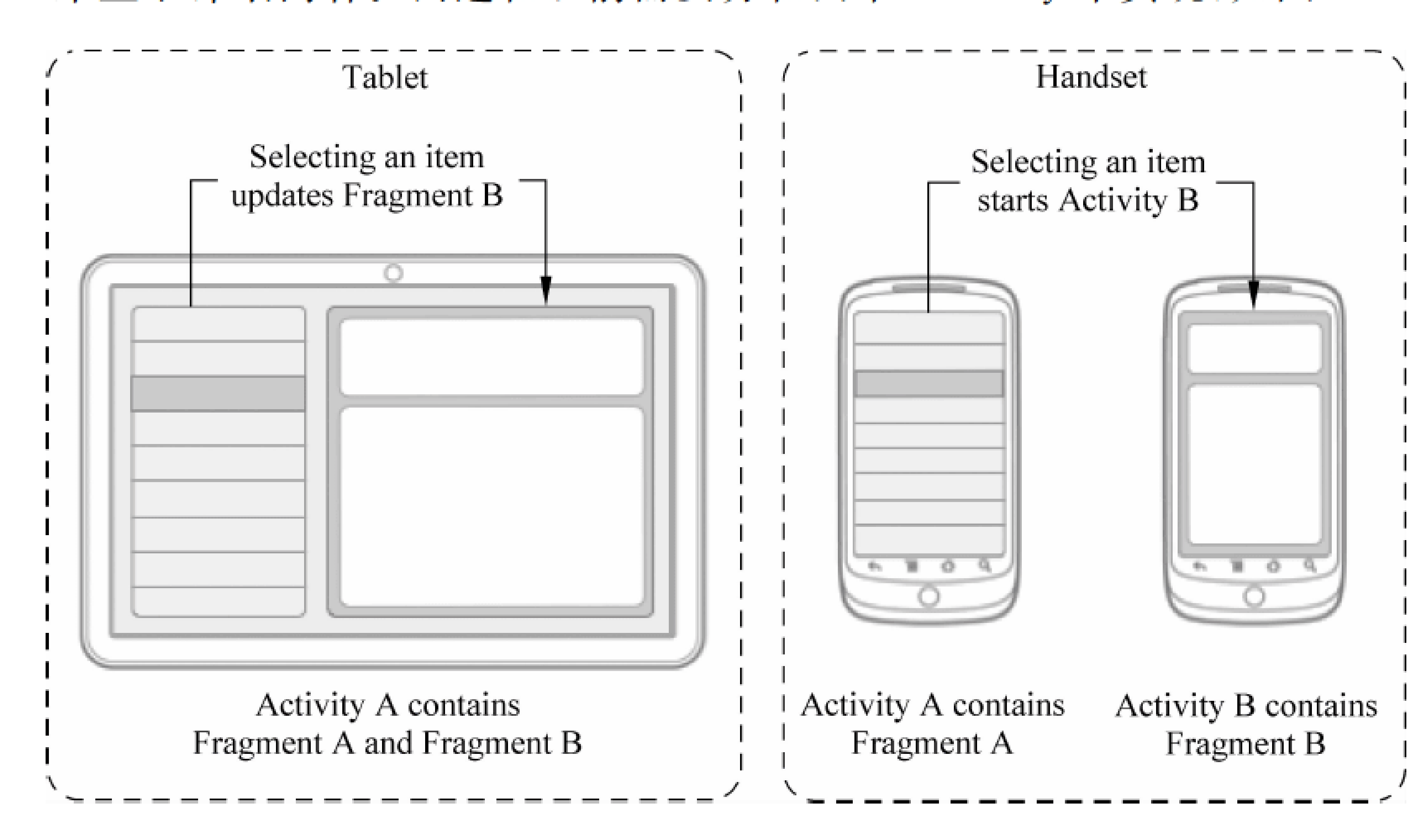


图 5.28 Fragment 与 Activity

5.5.1 创建并使用 Fragment

Fragment 直译过来就是片段的意思，所以它无法独立使用，必须放在 Activity 中，并受到 Activity 的影响。一个 Activity 可以拥有多个 Fragment，这些 Fragment 可以进行不同的操作，它们相互独立，也可以借助所在的 Activity 进行通信，这种机制大大增加了平板电脑和大屏幕手机上 UI 设计的灵活性。

Fragment 在 Activity 中运行时，拥有自己的生命周期，单独处理自己的输入。但是，Fragment 必须依赖于 Activity 存在，不能作为 Intent 的目标。Activity 可以加载或者移除某个 Fragment 模块。

Fragment 在 Android 的两个包中都有声明，一个是 android.app.Fragment 包，另一个是 android.support.v4.app.Fragment 包。前者是最原始的 Fragment，只支持 android 3.x 以上的设备。后者是 Google 的 android-support-v4.jar 包带的，可以让较低版本的操作系统使用一些高级 API 中的功能。

创建 Fragment 与创建 Activity 基本类似，需要继承 Fragment 类，并实现相应的方法。除了 Fragment 类之外，Android 还提供了 DialogFragment、ListFragment 和 PreferenceFragment。

其中,DialogFragment 是对话框式的 Fragment;ListFragment 类似于 ListActivity 的效果,直接具备 ListView 的功能,并且还提供了 ListActivity 类似的 onListItemCLick 和 setListAdapter 等功能;PreferenceFragment 类似于 Preference Activity,可以创建类似 iPad 的设置界面。上述 Fragment 在实现时,通常需要实现以下三个方法。

(1) onCreate,创建 Fragment 对象时调用,可以初始化 Fragment 中的控件,与 Activity 的 onCreate 方法类似。

(2) onCreateView,绘制用户界面的方法,该方法必须返回要创建的 Fragment 视图 UI 控件。如果不需要提供 Fragment 界面,则可以返回 null。

(3) onPause,当用户离开这个 Fragment 的时候调用,在该方法中可以进行数据的持久化处理。

将 Fragment 加入到 Activity 中有两种方法,一种方法是在 Activity 的 layout 中使用标签<fragment>声明(如代码 05-26 所示);另一种方法是在代码中把它加入到一个指定的 ViewGroup 中。

代码 05-26

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "horizontal"
    android:layout_marginLeft = "8dp"
    android:layout_marginRight = "8dp"
    android:divider = "?android:attr/dividerHorizontal"
    android:showDividers = "middle"
    >
    <fragment
        android:name = "com.freshen.code.FriendListFragment"
        android:id = "@ + id/mainlayout_list"
        android:layout_width = "0dp"
        android:layout_height = "match_parent"
        android:layout_weight = "1.0"
        />
    <FrameLayout
        android:id = "@ + id/mainlayout_detail"
        android:layout_width = "0dp"
        android:layout_height = "match_parent"
        android:layout_weight = "3.0"
        />
</LinearLayout>
```

新建项目 part05_3,演示类似于新闻浏览功能的实现过程,该项目在 Activity 的左侧是一个联系人列表,由自定义的 ListFragment 实现(如代码 05-27 所示),右侧是联系人详细信息区域,由自定义的 Fragment 实现(如代码 05-28 所示)。

代码 05-27

```
/*
 * 用于显示好友列表的 Fragment
 **/
public class FriendListFragment extends ListFragment {
    //声明对 Activity 的引用
    CallBack callBackActivity;
    public interface CallBack{
        public void onItemSelected(int index);
    }
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        if(!(activity instanceof CallBack)){
             throw new IllegalStateException("FriendListFragment 所在的 Activity 必须实现
CallBack 接口");
        }
        callBackActivity = (CallBack) activity;
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ArrayAdapter aa  = new ArrayAdapter(getActivity(),android.R.layout.simple_list_item
_activated_1,
                android.R.id.text1,FriendsData.fdata_list);
        setListAdapter(aa);
        Log.i("Tag", "FriendListFragment onCreate");
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i("Tag", "FriendListFragment onDestroy");
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);
        Log.i("Tag", "选择元素下标 " + position);
        callBackActivity.onItemSelected(position);
    }
}
```

代码 05-28

```
/*
 * 用于显示好友详细信息的 Fragment
 **/
public class FriendDetailFragment extends Fragment {
    Friend fr = null;              //实体类
    //创建 FriendFragment 的方法
```

```
    public static FriendDetailFragment newInstance(int id){
        FriendDetailFragment ff = new FriendDetailFragment();
        Bundle args = new Bundle();
        args.putInt("id", id);
        ff.setArguments(args);
        return ff;
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i("Tag", "FriendDetailFragment onCreate");
        if(getArguments()!= null&&getArguments().containsKey("id")){
            int id = getArguments().getInt("id");
            fr = FriendsData.fdata_map.get(id);
        }
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.frdetaillayout, null);
        if(fr!= null){
            TextView et1 = (TextView) view.findViewById(R.id.fd_np);
            et1.setText(fr.getName());
            TextView et2 = (TextView) view.findViewById(R.id.fd_msg);
            et2.setText(fr.getMsg());
            Log.i("Tag", "更新 view" + fr.getMsg());
        }
        return view;
    }
    @Override
    public void onDestroyView() {
        super.onDestroyView();
        fr = null;
        Log.i("Tag", "FriendDetailFragment onDestroy");
    }
}
```

上述代码分别自定义 Fragment，用于不同的功能。实体类 Friend 的详细信息可以查阅随书配套资料 part05_3。MainActivity 的功能如代码 05-29 所示。

📖 代码 05-29

```
public class MainActivity extends Activity implements CallBack{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mainlayout);
    }
    //实现 FriendListFragment 中的接口，用于回调，方便 Fragment 与 Activity 通信
    @Override
```

```
        public void onItemSelected(int index) {
            Bundle b = new Bundle();
            b.putInt("id", index);
            Log.i("Tag", "MainActivity bundle 数据 id = " + index);
            //根据回调方法传回的数据,创建详细信息 Fragment
            FriendDetailFragment fdf = FriendDetailFragment.newInstance(index);
            //更换成新的 Fragment
            getFragmentManager().beginTransaction().
                replace(R.id.mainlayout_detail, fdf).commit();
        }
    }
```

该项目的运行效果如图 5.29 所示。MainActivity 的布局文件中含有一个 ListFragment,用于显示朋友列表。在该 Activity 中实现 onCreate 方法,用于初始化数据,由于是继承了 ListFragment,所以可以直接设置数据适配器。在构建数据适配器时,使用的参数为(getActivity(),android. R. layout. simple_list_item_activated_1,android. R. id. text1,FriendsData. fdata_list)。参数的含义在第 4 章已经介绍过,此处 getActivity()方法是 Fragment 特有的,用于获取所处在的 Activity 对象;FriendsData. fdata_list 是用于模拟朋友列表数据的 List 集合。



图 5.29 Fragment 运行效果

在 FriendListFragment 中特意声明了接口 CallBack,实现与 Activity 通信。只有 Activity 实现该接口,那么在 FriendListFragment 中就可以通过方法的回调,将列表中被选中的数据传给 Activity。具体是在监听器方法 onListItemClick()中实现的。

MainActivity 已经实现了上述接口,可以得到 FriendListFragment 中所选中的数据,然后使用该数据创建 FriendDetailFragment 对象,并更新布局文件中 id 为 mainlayout_detail 的区域(实际是一个 FrameLayout)。

在 FriendDetailFragment 中实现方法 onCreateView(),对朋友详细信息进行了布局,使用方法 inflater. inflate(R. layout. frdetaillayout, null)实现,并根据传入的数据确定内容,随后返回该视图。

5.5.2 Fragment 生命周期

每一个 Fragment 对象都有自己的一套生命周期回调方法和处理用户输入事件的方法。因为 Fragment 必须放在 Acitivity 中才能使用，所以 Fragment 的生命周期和它所在的 Activity 是密切相关的。

如果 Activity 是暂停状态，其中所有的 Fragment 都是暂停状态；如果 Activity 是 Stop 状态，这个 Activity 中所有的 Fragment 都不能被启动；如果 Activity 被销毁，那么它其中的所有 Fragment 都会被销毁。

但是，当 Activity 在活动状态时，可以独立控制 Fragment 的生命周期，如图 5.30 所示，也可以进行添加或删除 Fragment 操作。

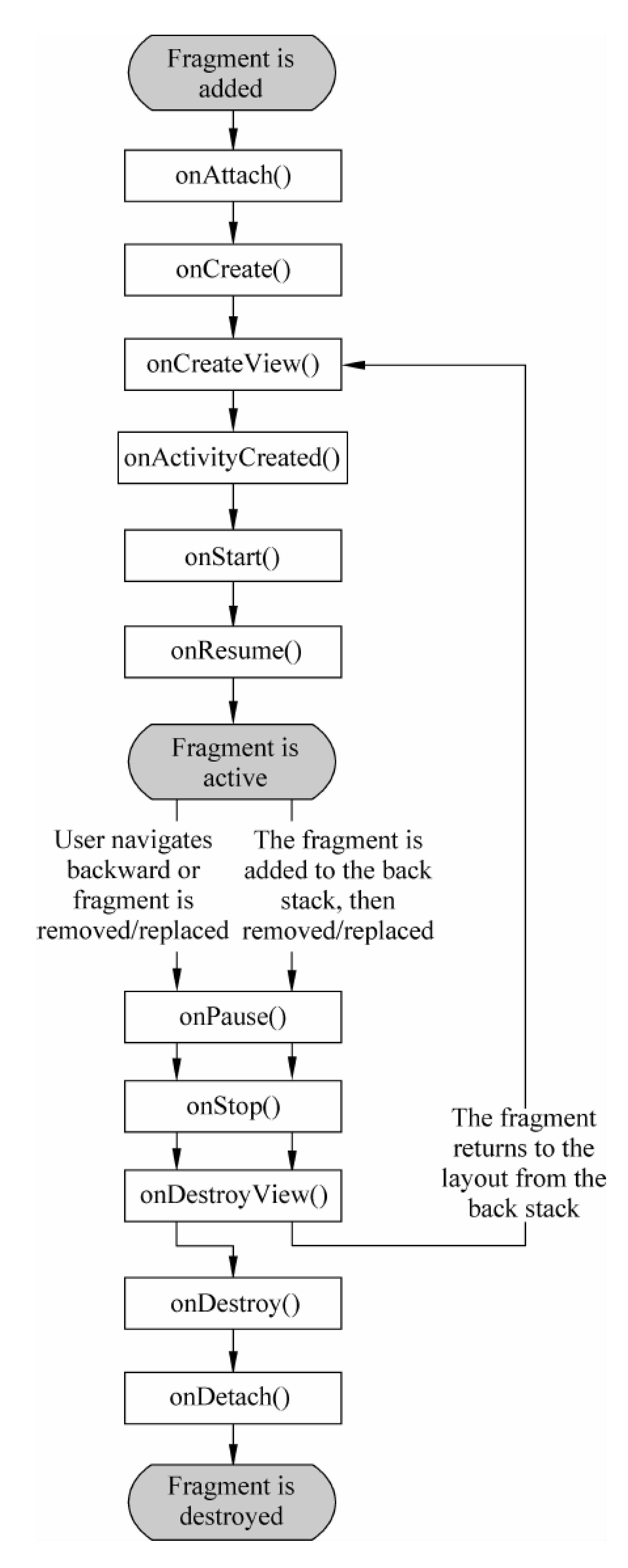


图 5.30 Fragment 生命周期

(1) onAttach()，当 Fragment 被添加到 Activity 中时执行，该方法只会执行一次。

(2) onCreate()，Fragment 创建时执行，该方法只会执行一次，类似于 Activity 的 onCreate 方法。

(3) onCreateView()，用于绘制 Fragment 的视图界面，Fragment 显示的界面是该方法返回的 View。

(4) onStart()，Fragment 启动阶段执行。

(5) onResume()，onStart 执行后，立即执行 onResume，与 Activity 中的 onResume 类似。

(6) onPause()，Fragment 失去焦点时执行，与 Activity 中 onPause 类似。

(7) onStop()，Fragment 不可见时执行，与 Activity 中的 onStop 类似。

(8) onDestroyView()，销毁 Fragment 所包含的 View 控件。

(9) onDestroy()，Fragment 销毁时执行该方法，与 Activity 的 onDestroy 类似，只会执行一次。

(10) onDetach()，Fragment 从 Activity 中移除时执行，只会执行一次。

由于Fragment的生命周期与Activity比较类似,此处不再使用单独的项目演示上述方法的使用,读者可以自己尝试各生命周期方法执行的时机。

5.5.3 管理Fragment

Fragment无法独立使用,需要作为Activity的组成部分;Activity中可以含有多个Fragment,并可以删除或添加Fragment。所以,这二者的通信和交互就显得比较重要。

获取Fragment所处的Activity,可以通过Fragment类中的方法getActivity()实现。

相反地,在Activity中,可以通过方法getFragmentManager(API 11之后)获取FragmentManager对象。然后通过FragmentManager的方法findFragmentById(int id)或findFragmentByTag(String tag),就可以得到对应的Fragment对象。

FragmentManager类中提供方法beginTransaction(),可以获取FragmentTransaction对象。FragmentTransaction提供了诸如add、hide、remove、replace、show等方法,用于添加、隐藏、移除、替换、显示指定的Fragment。并可以指定相应的动画效果,这些编辑将在commit()方法执行后生效。

使用FragmentManager和FragmentTransaction,就可以实现对Fragment的管理。这两个类都位于android.app包中。

习　题

1. MenuItem与SubMenu有何区别?
2. 一个标准Dialog的组成部分有哪些?
3. android:showAsAction属性的作用什么?可以取哪些值?各有什么特点?
4. 与Fragment生命周期相关的方法有哪些?
5. Fragment之间如何通信?



CHAPTER 6

第6章

Android四大组件

本章主要内容：

- Activity；
- Service；
- BroadcastReceiver；
- ContentProvider；
- Intent 与 Intent Filter。

在开发 Android 应用程序时，一般都会用到 Activity、Service、BroadcastReceiver 和 ContentProvider，但不是必须同时用到这 4 个组件，前面编写的测试程序只涉及 Activity。在此将其称为四大组件，本章主要介绍上述四大组件。

Activity 作为应用程序的交互界面存在，需要呈现给使用者；Service 相当于后台运行的 Activity，用户无法与其直接交互；BroadcastReceiver 用于接收广播；ContentProvider 支持在多个应用中存储和读取数据，相当于数据库。

6.1 Activity

Activity 是软件开发过程中使用最为频繁的一个组件，在第 2 章中就对 Activity 进行了介绍，并在前面几章的测试中反复使用了 Activity。如果把手机屏幕看作浏览器，Activity 就相当于其中的页面，Activity 中可以呈现不同的 View(视图)。一般情况下应用程序由多个 Activity 组成，它们之间可以通过 Intent 相互跳转，如触发按钮单击、下拉列表选择、菜单单击等，在前面几章的学习中，已经接触过此类操作。

Activity 拥有很多生命周期方法，在不同阶段会执行与之对应的方法。当打开一个新的 Activity 时，之前的 Activity 会被置为暂停状态，并且压入历史堆栈中。但按手机上的 Back 键时，返回到以前打开过的 Activity。Activity 对象是由 Android 系统进行维护的。该部分内容可以查阅第 2 章的相关知识。

6.2 Service

Service 正如其名，是一种服务性质的组件。不能被用户所见，只能在后台运行，但可以和其他组件进行交互。Service 间接继承 Context，位于 android. app 包中，与 Activity 的级别差不多，不能依靠自己运行，必须通过某个 Activity 或者其他的 Context 对象来调用。

Service 可以看作是没有交互、不可见的 Activity，可以在后台有规律地执行某些操作，也可以为其他对象提供接口，以便于在应用程序中调用。Service 组件与其他的应用程序对象一样，是运行在启动它的主线程中，如果 Service 中有比较耗时的操作，会影响到主线程，甚至阻塞主线程。对于比较耗时的操作，最好启动新的线程。Service 作为没有界面的长生命周期的组件，常常被应用于媒体播放、检测 SD 卡上的文件变化、记录用户地理位置等场合。

6.2.1 新建 Service

根据 Service 启动方式与运行的不同，可以分为 Local Service(本地服务，一般用于应用程序内部)和 Remote Service(远程服务，一般用于 Android 系统内部的应用程序之间)。

Local Service 通过调用 Context. startService()启动，调用 Context. stopService()结束。也可以通过调用 Service. stopSelf()或 Service. stopSelfResult()来停止。对于同一种 Service，无论调用了多少次 startService()方法，只需要调用一次 stopService()来停止服务。如果需要与 Service 交互，可以使用 Context. bindService()方法建立绑定，使用 Context. unbindService()关闭绑定。

Remote Service 可以通过自己定义并暴露出来的接口进行程序操作。应用程序建立一个到 Service 的连接，并通过那个连接来调用服务。连接以调用 Context. bindService()方法建立，Context. unbindService()用于关闭。多个应用程序可以绑定至同一个 Service。

新建类 MyService，继承 Service(android. app 包中)，参考代码 06-1，完整代码请查阅随书配套资料 Part06 项目。重写代码中给出的方法，其中 onBind 方法是 Service 类的抽象方法，必须实现，稍后会介绍如何使用该方法。

代码 06-1

```
public class MyService extends Service {
    //该方法是 Service 类中定义的抽象方法,必须实现
    //用于获取该 Service 对象,如果返回 null,则意味着该 Service 不支持 bind
    @Override
    public IBinder onBind(Intent arg0) {
        Log.i("SINFO", "onBind running.");
        return null;
    }
    @Override
    public void onCreate() {
        Log.i("SINFO", "onCreate running.");
        super.onCreate();
```

```
        }
        @Override
        public void onDestroy() {
            Log.i("SINFO", "onDestroy running.");
            super.onDestroy();
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            Log.i("SINFO", "onStartCommand running.");
            return super.onStartCommand(intent, flags, startId);
        }
        @Override
        public boolean onUnbind(Intent intent) {
            Log.i("SINFO", "onUnbind running.");
            return super.onUnbind(intent);
        }
    }
```

与 Activity 中的生命周期方法类似，上述 5 个方法都是 Service 的生命周期方法。新建 Service 对象，需要重写这 5 个方法。为测试启动与停止该 Service，在 MainActivity.java 所采用的布局视图中添加按钮控件，详细代码参考 Mainactivity.java 文件和 main.xml 布局文件。

在运行该测试程序之前，需要在 Manifest 文件中配置 Service 的相关信息。在 application 标签中添加<service android:name=".MyService" android:enabled="true"></service>，指明新建的 Service 类名，并启用该 Service。运行 MainActivity，单击 Start Service 按钮，Service 对象的方法 onCreate 和 onStartCommand 执行，继续单击该按钮，只有 onStartCommand 方法执行。单击 Stop Service 按钮，onDestroy 方法执行。onBind 方法和 onUnbind 方法都没有执行。Service 启动过程记录如图 6.1 所示。

Level	Time	PID	Application	Tag	Text
I	05-16...	382	com.freshen.service	SINFO	onCreate running.
I	05-16...	382	com.freshen.service	SINFO	onStartCommand running.
I	05-16...	382	com.freshen.service	SINFO	onStartCommand running.
I	05-16...	382	com.freshen.service	SINFO	onStartCommand running.
I	05-16...	382	com.freshen.service	SINFO	onDestroy running.
I	05-16...	382	com.freshen.service	SINFO	onCreate running.
I	05-16...	382	com.freshen.service	SINFO	onStartCommand running.

图 6.1 Service 启动过程记录

在 MainActivity 中单击 Start Service 按钮，执行 startService 方法，启动服务；单击 Stop Service 按钮，执行 stopService 方法，停止服务。这两个方法都是 Activity 从 ContextWrapper 类中继承的。

上面的代码演示不需要和 Activity 交互的本地服务的使用，下面实现在 Activity 中获取 Service 对象，并与其交互，调用 Service 中的相应方法。

修改 MyService，添加内部类继承 Binder，用于向绑定该 Service 的应用程序提供该 Service 对象，参考代码 06-2。当其他应用程序，比如 Activity 使用 bindService 方法，绑定

Service 时，并不能直接获取到 Service 对象，需要借助 ServiceConnection 对象。

代码 06-2

```
@Override
public IBinder onBind(Intent arg0) {
    Log.i("SINFO", "onBind running.");
    return new MyIBinder();
}
//自定义类继承 Binder,供 onBind 方法使用
class MyIBinder extends Binder{
    MyService getService(){
        return MyService.this;
    }
}
```

修改 MainActivity，添加 ServiceConnection 属性，并创建对象，参考代码 06-3。onServiceConnected 方法会在绑定成功时执行，其中参数 IBinder 就是 Service 对象中 onBind 方法返回的对象。获取到 Service 对象后，就可以调用其中的方法了。

代码 06-3

```
//用于连接 Activity 和 Service
ServiceConnection sc = new ServiceConnection(){
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //通过 IBinder 对象获取 Service 对象
            MyService ms = ((MyService.MyIBinder)service).getService();
            //此处调用 Service 对象中的相应方法即可
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
        }}
    …
            Intent intent = new Intent(this,MyService.class);
            //绑定 Service
            this.bindService(intent, sc,BIND_AUTO_CREATE);
    …
            //解除绑定
            this.unbindService(sc);
```

绑定 Service 采用的方法是 bindService，第一个参数指明绑定的 Service，第二个参数是 ServiceConnection 对象，可以看作是 Activity 与 Service 的桥梁，第三个参数是绑定时的方式。解除绑定使用的方法是 unbindService，需要传入的参数是 ServiceConnection。如果已经重复执行解除绑定操作，会抛出异常信息，为避免重复解除绑定，可以设置一个 boolean 类型的变量，作为解除的标志。代码中并未实现，感兴趣的读者可以自行尝试。

第二种启动 Service 的方法是通过 ServiceConnection 对象，将 Activity 与 Service 联系起来。这种方式会让启动的服务与调用者（如 Activity）绑定，当调用者关闭时服务也就终止了。运行测试程序，通过 Bind Service 按钮和 Unbind Service 按钮，测试启动服务和关闭

服务，输出结果如图 6.2 所示，Service 对象的 onStartCommand 方法不会执行。由于 Service 已经与 Activity 绑定，当 Activity 退出时，也会执行 onUnbind 和 onDestroy 方法。

Level	Time	PID	Application	Tag	Text
I	05-16...	443	com.freshen.service	SINFO	onCreate running.
I	05-16...	443	com.freshen.service	SINFO	onBind running.
I	05-16...	443	com.freshen.service	SINFO	onUnbind running.
I	05-16...	443	com.freshen.service	SINFO	onDestroy running.

图 6.2 Service 绑定过程记录

6.2.2 Service 的生命周期

启动一个 Service 有两种方式：调用 Context.startService() 或 Context.bindService()。何种启动方式决定了 Service 应该执行哪些生命周期方法。Service 的生命周期比 Activity 的生命周期要简单，由于 Service 是运行在后台，使用者无法感知它的存在，所以了解 Service 的生命周期就显得比较重要了。

在同一个应用程序任何位置调用 startService()方法都能启动 Service。系统回调 Service 类的 onCreate()方法以及 onStart()方法。这种方式启动的 Service 会一直运行在后台，直到调用 Context.stopService()或者 selfStop()方法。如果一个 Service 已经被启动，其他代码再试图调用 startService()方法启动 Service，将不执行 onCreate()，而是重新执行一次 onStart()方法。

另一种 bindService()方法也可以启动 Service，并把这个 Service 和调用 Service 的应用程序绑定起来，如果调用 Service 的应用程序类被销毁，Service 也将被销毁。使用这个方法的一个好处是，bindService()方法执行后 Service 会回调 onBind()方法，可以从这里返回一个实现了 IBinder 接口的类，如代码 06-2，在应用程序中就可以通过这个类和 Service 通信了，比如得到 Service 运行的状态或其他操作。如果 Service 没有运行，使用这个方法启动 Service 会调用 onCreate()方法，但不会调用 onStart()，两种启动方式的生命周期如图 6.3 所示。

Service 的这两种启动方式并不是相互独立的，有可能出现交叉调用的情况。如果出现这种情况，需要按照启动的顺序依次调用与启动对应的结束方式。例如，先调用 startService()，然后再调用 bindService()，启动并绑定 Service 可以达成既保持和 Service 的通信，又使得 Service 不会随着 Activity 的退出而退出。当不需要绑定时，先调用 unbindService()方法，此时 Service 会执行 onUnbind()，但不会把这个 Service 销毁。只有再调用 stopService()时，Service 才会销毁。

Service 中经常使用的与生命周期相关的方法，以及方法的描述如表 6.1 所示。

表 6.1 Service 类中生命周期方法

方法名	作　　用	备　　注
onCreate()	该方法在 Service 被创建时调用，且只会被调用一次，无论调用多少次 startService()或 bindService()方法，Service 只被创建一次	

续表

方法名	作　　用	备　　注
onDestroy()	该方法在 Service 被终止时调用	
onStart()	只有采用 Context. startService()方法启动 Service 时才会调用该方法。多次调用 startService()方法不会多次创建 Service,但会多次调用 onStart() 方法	Context. startService()启动方式会调用该方法
onBind()	只有采用 Context. bindService()方法启动 Service 时才会调用该方法。该方法在应用程序与 Service 绑定时被调用,多次调用 Context. bindService()方法并不会导致该方法被多次调用	Context. bindService()启动方法会调用该方法
onUnbind()	只有采用 Context. UnbindService()方法解除绑定 Service 时才会调用该方法	Context. bindService()启动方法会调用该方法

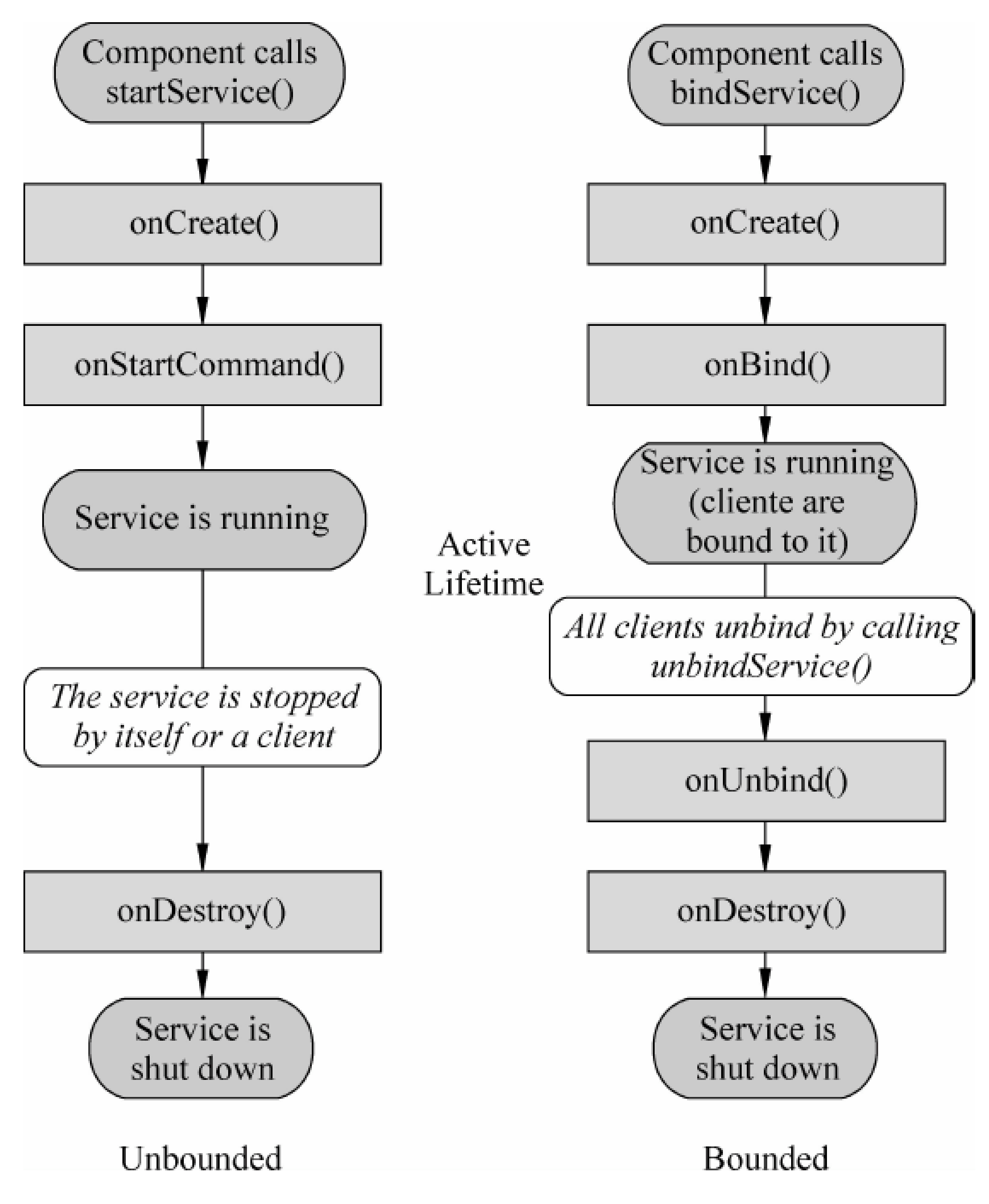


图 6.3　Service 生命周期

6.2.3　Local Service 和 Remote Service

Service 启动后会运行在主线程中,如果有耗时操作,可以在 Service 中开启线程。根据所使用区域不同,Service 分为 Local Service(本地服务)和 Remote Service(远程服务)。

(1) Local Service:用于应用程序内部,当不需要与 Service 交互时,可以采用 Context. startService()方式启动;需要与 Service 交互时,采用 Context. bindService()启动。这种

Service 使用简单，应用场合比较多，上面介绍的就是这种 Service。

（2）Remote Service：这种 Service 可以让其他应用程序访问该服务（Android 各应用程序都是互相独立的，数据独享），实际上就是可以进程间通信（RPC），这时需要使用 Android 提供的接口描述语言（AIDL）来定义远程服务的接口。这种 Service 比较复杂，特殊场合需要用到。

1. Local Service

所谓本地服务是相对于一个应用程序而言的，当 Service 作为某一个应用程序的组成部分，用于运行在后台，处理相应功能，就可以看作是一个 Local Service。下面通过一个简易播放器，讲解 Local Service 的使用。

第一，新建布局文件 playerlayout.xml，作为播放器的操作界面。采用 TableLayout 布局，放入三个 ImageButton，用于控制播放、暂停和停止，如代码 06-4 所示。

代码 06-4

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TableLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:stretchColumns="*"
        android:padding="1dip"
        android:background="#cccccc" >
        <TableRow>
            <ImageButton android:id="@+id/ib_play"
                android:src="@drawable/play"
                android:scaleType="fitCenter"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
            <ImageButton android:id="@+id/ib_pause"
                android:src="@drawable/pause"
                android:scaleType="fitCenter"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
            <ImageButton android:id="@+id/ib_stop"
                android:src="@drawable/stop"
                android:scaleType="fitCenter"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
        </TableRow>
```

```
    </TableLayout>
</LinearLayout>
```

第二，新建 PlayerActivity.java，作为播放器的主 Activity。在此启动 Service，让播放工作在 Service 中进行，这样即使 Activity 退出，播放会依然继续。在启动 Service 时需要向 Service 中传入数据，表示当前的操作，可以借助 Intent 实现(稍后会详细解决 Intent 的使用)。如果需要退出播放器，可以再增加一个按钮，当触发时，向 Service 传递结束信息即可，这个功能读者可以自己实现。主要功能如代码 06-5 所示。

代码 06-5

```
public class PlayerActivity extends Activity implements OnClickListener {
    ImageButton ib1, ib2, ib3;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto - generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.playerlayout);
        ib1 = (ImageButton) findViewById(R.id.ib_play);
        ib1.setOnClickListener(this);
        ib2 = (ImageButton) findViewById(R.id.ib_pause);
        ib2.setOnClickListener(this);
        ib3 = (ImageButton) findViewById(R.id.ib_stop);
        ib3.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        intent = new Intent(this, PlayerService.class) ;
        switch (v.getId()){
        case R.id.ib_play:
            intent.putExtra("op", 1);
            break;
        case R.id.ib_pause:
            intent.putExtra("op", 2);
            break;
        case R.id.ib_stop:
            intent.putExtra("op", 3);
            break;
        }
        //启动 Service
        startService(intent);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
    }
}
```

第三，新建 Service。播放音乐可以使用 MediaPlayer 类，在后续章节中会有介绍。本例中是播放 SD 卡中的音乐文件，设置播放文件可以采用 player.setDataSource("/sdcard/music/my love.mp3")。onCreate 方法在 Service 的生命周期中只会执行一次，在此可以初始化播放器。onStartCommand 方法会被多次调用，可以在此判断用户的操作，执行相应方法，对于 MediaPlayer 对象的操作现在可以不关注，如代码 06-6 所示。

代码 06-6

```
public class PlayerService extends Service {
    MediaPlayer player;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        //指定播放音乐文件
        player = new MediaPlayer();
        try {
            player.setDataSource("/sdcard/music/my love.mp3");
            player.prepare();
        } catch (IllegalArgumentException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        } catch (IllegalStateException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        }
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        //停止播放,释放资源
        if(player!= null){
            player.stop();
            player.release();
        }
        Log.i("INFO", "Service destroy!");
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("INFO", "onStartCommand" );
        if(intent!= null){
            int op  = intent.getIntExtra("op", 0);
```

```
                Log.i("INFO", "op = " + op);
                if(op == 1){
                    play();
                }else if(op == 2){
                    pause();
                }else if(op == 3){
                    stop();
                }
            }
            return super.onStartCommand(intent, flags, startId);
        }
        //播放控制的方法
        private void stop() {
            Log.i("INFO", "stop op");
            if(player!= null){
                player.stop();
                try {
                    player.prepare();
                } catch (IllegalStateException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                }
            }
        }
        private void pause() {
            Log.i("INFO", "pause op");
            if(player!= null&&player.isPlaying())player.pause();
        }
        private void play() {
            Log.i("INFO", "play op");
            if(!player.isPlaying())player.start();
        }
        @Override
        public boolean onUnbind(Intent intent) {
            return super.onUnbind(intent);
        }
    }
```

第四,配置 Activity 和 Service,具体的配置信息和完整的代码可以查看随书配套资料 Part06 项目。上述代码的运行效果如图 6.4 所示。由于在 Activity 中只是启动了 Service,并没有绑定,所以当 Activity 退出时,播放器依然工作。

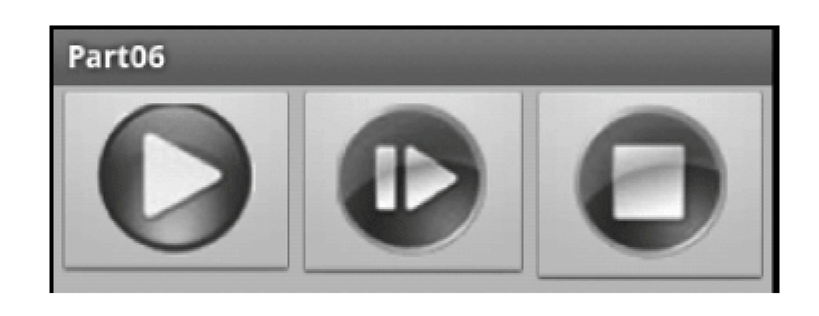


图 6.4 简易播放器界面

2. Remote Service

在 Android 系统中，每个应用程序都是运行在自己独立的进程。一般情况下，一个进程不能访问另一个进程的内存空间。在某些特定应用中，需要访问另一个应用程序时，需要将对象分解成操作系统可以理解的基本单元，Android 采用 AIDL（Android Interface Definition Language）来完成这项工作。AIDL 是一种接口定义语言，用于在两个进程之间通信。按照以下步骤使用 AIDL 实现两个进程间的通信。

第一，在代码包中创建 MessageInterface. aidl 文件，这只是一种普通的文件，只是后缀名为. aidl。打开该文件，定义接口如代码 06-7 所示。在该文件中定义接口与普通 Java 接口是一样的，只是在 aidl 文件中出现的复杂类型需要显式调用 import，即使是在同一个包中。Java 中的 8 种基本类型、String、CharSequence 和集合接口类型中的 List 和 Map 必须要 import。

代码 06-7

```
package com.freshen.service;
interface MessageInterface{
    String getMessage();
}
```

当 aidl 文件创建后，该项目的 gen 包中会自动生成一个同名的 Java 类，与 R 类在同一个包中，感兴趣的读者可以打开看看。该类中的代码比较繁杂，其中内部类 Stub 是需要关注的，Stub 继承了 Binder，并实现 AIDL 中定义的接口。除此之外还需要关注一个方法，该方法与在 aidl 文件中定义的同名。

第二，新建 Service 类。重写生命周期的相关方法，新建内部类 MessageInterfaceImp，继承 Stub。该内部类的作用与前面介绍的采用 bind 方法启动 Service 时的内部类一致，只是实现过程不同了。在此用一个随机数生成器返回随机数，当其他应用程序调用该 Service 的方法时，会提供一个随机数字符串。

代码 06-8

```
public class MessageService extends Service {
    static final String TAG = "Tag";
    //内部类继承 AIDL 自动生成类的内部类 Stub,并实现 AIDL 中声明的方法
    private class MessageInterfaceImp extends MessageInterface.Stub{
        Random r = new Random();
        @Override
        public String getMessage() throws RemoteException {
            return "msg " + r.nextInt(100);
        }
    }
    @Override
    public IBinder onBind(Intent arg0) {
        Log.i(TAG, "onBind run.");
        //返回 AIDL 接口实现
```

```
            return new MessageInterfaceImp();
        }
        @Override
        public void onCreate() {
            Log.i(TAG, "onCreate run.");
            super.onCreate();
        }
        @Override
        public void onDestroy() {
            Log.i(TAG, "onDestroy run.");
            super.onDestroy();
        }
        @Override
        public boolean onUnbind(Intent intent) {
            Log.i(TAG, "onUnBind run.");
            return super.onUnbind(intent);
        }
    }
```

在 manifest 文件中配置该 Service。与前面配置 Service 不同的是，该 Service 需要添加属性 android:process，指明该 Service 在启动时会在另一个线程中，如代码 06-9 所示。如果该属性的值没有，则在 Service 启动时会创建一个全局进程，不同的应用程序共享该进程。这样配置后，Service 和测试 Activity 将不在同一个进程，可以模拟远程调用。当然，也可以让该 Service 充当服务器端，再重新创建一个客户端应用程序，将 aidl 文件复制到客户端中，这样 Service 和调用者也会在两个进程中。无论何种方式，最终的目的是达成 Service 和调用者不在同一个进程的效果。

📖 代码 06-9

```
<!-- 运行时会开启新的进程 -->
        <service android:name=".MessageService" android:enabled="true" android:process
="：remote">
            <intent-filter>
                <action android:name="com.freshen.remoteservice.MessageService"/>
            </intent-filter>
        </service>
```

第三，新建客户端应用程序，比如使用 Activity 来测试。在项目中创建 MessageActivity.java，如代码 06-10 所示，布局文件为 messagelayout.xml。完整代码请查看 Part06 项目。在绑定 Service 时需要用到 ServiceConnection，用法与前面介绍的类似。与 Service 交互需要借助 aidl 文件中声明的方法，获取该接口的实现对象是借助 Stub 类完成的。

📖 代码 06-10

```
public class MessageActivity extends Activity implements OnClickListener{
    static final String TAG = "Tag";
    private TextView tv;
```

```
    private Button b1,b2;
    MessageInterface msgif;          //AIDL中定义的
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.messagelayout);
        tv = (TextView) findViewById(R.id.msg_tv1);
        b1 = (Button) findViewById(R.id.msg_bt1);
        b2 = (Button) findViewById(R.id.msg_bt2);
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }
    //Service连接使用
    private ServiceConnection sc = new ServiceConnection(){
        @Override
        public void onServiceConnected(ComponentName arg0, IBinder arg1) {
            Log.i(TAG, "sc.onServiceConnected!");
            //得到AIDL中声明的对象
            msgif = MessageInterface.Stub.asInterface(arg1);
        }
        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            Log.i(TAG, "sc.onServiceDisconnected!");
        }
    };
    @Override
    public void onClick(View v) {
        switch(v.getId()){
        case R.id.msg_bt1:
            bindService();
            break;
        case R.id.msg_bt2:
            getMessage();
            break;
        }
    }
    public void bindService(){
        Intent intent  = new Intent(this,MessageService.class);
        startService(intent);
        bindService(intent, sc, BIND_AUTO_CREATE);
    }
    public void getMessage(){
        Log.i(TAG, "getMessage");
        try {
            tv.append(msgif.getMessage() + "\n");
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

上述代码的运行效果如图 6.5 所示。从 PID(进程编号)和 Application(应用程序)都可以看出 Service 和 Activity 是在不同进程中运行的。这种远程调用的 Service 常用于获取 GPS 信息、SD 卡变化、数据下载等功能。在使用 Service 组件时应该注意以下几点。

Level	Time	PID	Application	Tag	Text
I	05-23...	472	com.freshen.service:remote	Tag	onCreate run.
I	05-23...	472	com.freshen.service:remote	Tag	onBind run.
I	05-23...	462	com.freshen.service	Tag	sc.onServiceConnected!
I	05-23...	462	com.freshen.service	Tag	getMessage
I	05-23...	472	com.freshen.service:remote	Tag	return msg
I	05-23...	462	com.freshen.service	Tag	getMessage
I	05-23...	472	com.freshen.service:remote	Tag	return msg

图 6.5 Remote Service 运行记录

- Service 只有在初次创建时执行 onCreate 方法，如果在服务启动后再调用 startService，只会执行 onStartCommand 方法，执行相应功能的代码应放在合适的位置。
- 注意 Service 两种启动方式的不同，所执行的生命周期方法也不同。
- 要实现远程调用 Service，即 Service 和调用应用程序(也称客户端)不在同一进程中，需要借助 AIDL 描述语言，定义接口。
- 能用本地 Service 解决的功能，尽量采用这种方法。

6.3 BroadcastReceiver

Android 系统中的广播实质上指的是发送 Intent，用于在应用程序之间传送消息，但与 Activity 中使用的 Intent 是不同的。Activity 中的 Intent 是在前台执行的，广播中的 Intent 是后台执行，用户是无法感知的。BroadcastReceiver 是广播接收器，用于接收来自系统和应用程序中的广播。

广播的使用在 Android 应用程序开发中非常方便，只需要在特定位置等待广播的到来即可，例如，系统开机后会产生一条广播信息，接收到这条信息就可以实现开机便启动服务的功能；当网络连接改变时会产生一条广播信息，接收该广播信息就可以根据网络状态执行相应操作；当电池电量改变时会产生广播信息，接收该广播信息可以处理耗电应用程序等，诸如此类的广播非常普遍，而这些任务都可以由 BroadcastReceiver 来实现。

6.3.1 广播接收器的注册

广播可以来源于系统，称为系统广播，也可以在自己的应用程序中发出广播。发送广播是使用 android.content.ContextWrapper 中的方法，具体描述可以参考表 6.2。前面介绍的 Activity 和 Service 都继承自 ContextWrapper，因此可以在 Activity 或 Service 中发出广播。

表 6.2 常用发送广播的方法介绍

方 法 名	功 能
sendBroadcast（Intent intent）	发送普通广播，匹配的 BroadcastReceiver 都会接收到该广播，并执行 onReceiver 方法，如有多个 BroadcastReceiver 都匹配，则相应的先后顺序不确定
sendOrderedBroadcast（Intent intent，String receiverPermission）	发送有序广播，匹配的 BroadcastReceiver 会接收到广播，当多个接收器都匹配时，根据注册时 IntentFilter 的优先级顺序响应
sendStickyBroadcast（Intent intent）	发送黏性广播，已经注册的 BroadcastReceiver 匹配后，响应该广播；该广播的 Intent 会一直保持，在广播发出后，如有再注册的 BroadcastReceiver 也会接收到该条广播

发出广播就与各个无线电台发出无线电波一样，广播接收器就如同调频收音机。广播发出后，如果不是黏性广播会很快就消失（约 10s）。接收广播就需要注册 BroadcastReceiver，注册方式主要有静态注册和动态注册，两种注册方式区别较大，使用于不同的广播接收场合。

1. 静态注册

静态注册就是在 Manifest 文件中，使用<receiver>标签配置。配置时需要使用<intent-filter>指明所接收的广播，如同收音机指明接收的频段一样。静态注册的广播接收器是常驻型的，即使当应用程序关闭，如果有匹配的广播来到，接收器也会响应。如果此类配置的接收器较多，会给系统的运行增大负担，这类接收器多用于接收系统广播，如电话、短信等。

新建 Android 项目 part06_1，完整代码请参考随书配套资料。在 Activity 所使用的布局文件中添加按钮，用于发出广播。新建类 MyBroadcastReceiver 继承 BroadcastReceiver，重写 onReceive 方法（对于 BroadcastReceiver 来说，只需要关注该方法，其他方法都被 final 修饰）。

代码 06-11

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    static final String TAG = "Tag";
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取随 Intent 到达的数据
        String msg = intent.getStringExtra("msg");
        Log.i(TAG, "接收器收到信息 »" + msg);
    }
}
```

与 Activity、Service 组件一样，静态广播也需要在 Manifest 文件中配置，如代码 06-12 所示。action 指明该接收器将要响应的广播。在 Activity 中发送广播的代码参考代码 06-13，创建广播 Intent，并指明自定义动作，当广播接收器与此动作匹配时，就能收到该条广播。Intent 在 Activity、Service、BroadcastReceiver 中频繁出现，对于 Intent 的介绍在

6.5 节中进行。LogCat 输出的记录如图 6.6 所示。

📖 **代码 06-12**

```
<receiver android:name = ".MyBroadcastReceiver">
    <intent - filter >
        <action android:name = "com.freshen.code.MyBroadcastReceiver" />
        <category android:name = "android.intent.category.DEFAULT" />
    </intent - filter >
</receiver >
```

📖 **代码 06-13**

```
@Override
    public void onClick(View arg0) {
        //创建广播 Intent
        Intent intent  = new Intent("com.freshen.code.MyBroadcastReceiver");
        intent.putExtra("msg","广播信息 : 随机数 " + rand.nextInt(100));
        //发出广播
        //匹配 com.freshen.code.MyBroadcastReceiver 动作的广播接收器将接收到该广播
        sendBroadcast(intent);
        Log.i(TAG, "广播已发出");
    }
```

Level	Time	PID	Application	Tag	Text
I	05-25...	467	com.freshen.code	Tag	广播已发出
I	05-25...	467	com.freshen.code	Tag	接收器收到信息 》广播信息 : 随机数 13
I	05-25...	467	com.freshen.code	Tag	广播已发出
I	05-25...	467	com.freshen.code	Tag	接收器收到信息 》广播信息 : 随机数 24

图 6.6 接收器工作记录

2. 动态注册

动态注册广播接收器不需要借助配置文件，使用 ContextWrapper. registerReceiver (BroadcastReceiver receiver，IntentFilter filter)。该方法的两个参数含义如下。

(1) receiver：需要注册的广播接收器。

(2) filter：广播接收器需要匹配的动作。

修改 part06_1 项目中的 MainActivity 代码，添加三个按钮，用于注册、发出广播、取消注册，主要代码如代码 06-14 所示，完整代码请查看随书配套资料。

📖 **代码 06-14**

```
    …
@Override
public void onClick(View arg0) {
    if(arg0.getId() == R.id.bc_bt01){
        //
```

```
            Intent intent = new Intent("com.freshen.code.MyBroadcastReceiver");
            intent.putExtra("msg","广播信息 : 随机数 " + rand.nextInt(100));
            //发出广播
            //匹配 com.freshen.code.MyBroadcastReceiver 动作的广播接收器将接收到该广播
            sendBroadcast(intent);
            Log.i(TAG, "广播已发出");
        }
        if(arg0.getId() == R.id.bc_bt02){
            //注册广播接收器
            receiver2 = new MyBroadcastReceiver2();
            IntentFilter filter = new IntentFilter("com.freshen.code.MyBroadcastReceiver2");
            registerReceiver(receiver2, filter);      //执行注册
            Log.i(TAG, "广播接收器已注册");
        }else if(arg0.getId() == R.id.bc_bt03){
            //发送广播
            Intent intent = new Intent("com.freshen.code.MyBroadcastReceiver2");
            intent.putExtra("msg","广播信息 : 随机数 " + rand.nextInt(100));
            //发出广播
            //匹配 com.freshen.code.MyBroadcastReceiver 动作的广播接收器将接收到该广播
            sendBroadcast(intent);
            Log.i(TAG, "广播已发出");
        }else if(arg0.getId() == R.id.bc_bt04){
            //解除广播接收器
            if(receiver2!= null){
                unregisterReceiver(receiver2);       //取消注册
                receiver2 = null;
            }
            Log.i(TAG, "执行解除注册操作");
        }
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //解除广播接收器
        if(receiver2!= null)
            unregisterReceiver(receiver2);           //取消注册
    }
```

当单击“注册”按钮时,动态注册了广播接收器,并指明接收的广播。“发送”按钮的功能与静态注册广播接收器一致。“解除注册”按钮调用 unregisterReceiver 方法,解除注册的广播接收器。需要注意的是动态注册的广播接收器,在应用退出时需要解除注册,否则会抛出 receiver unregistered 异常。

代码 06-14 为了演示注册、发送广播、解除注册三个功能,直接将代码写在了监听器中,这样在开发中并不合理。根据应用的不同,需要开始注册和解除注册可以写在 Activity 或 Service 的相应生命周期方法中,比如在 onDestroy 方法中解除注册。上述代码的运行记录如图 6.7 所示。

当广播注册后,发送广播,接收器可以收到相应信息;当广播取消注册后,再发送广播,

Level	Time	PID	Application	Tag	Text
I	0...	560	com.freshen.code	Tag	广播接收器已注册
I	0...	560	com.freshen.code	Tag	广播已发出
I	0...	560	com.freshen.code	Tag	动态注册接收器，收到信息 》广播信息 ： 随机数 15
I	0...	560	com.freshen.code	Tag	广播已发出
I	0...	560	com.freshen.code	Tag	动态注册接收器，收到信息 》广播信息 ： 随机数 69
I	0...	560	com.freshen.code	Tag	执行解除注册操作
I	0...	560	com.freshen.code	Tag	广播已发出
I	0...	560	com.freshen.code	Tag	广播已发出

图 6.7 动态注册广播接收器记录

接收器没有响应。

动态注册的广播接收器无法驻留到系统中，当应用程序的生命周期结束后，广播接收也就随之消失。静态注册和动态注册两种方式在选择时应注意以下原则。

- 对于自己发送和接收的广播可以通过 registerReceive 动态注册，对于系统常用广播的接收通常用<receiver>标签注册。
- registerReceive 可以手动控制，适当地注册和取消注册能节省系统资源，<receiver>标签在系统开机后一直有效，动态注册能解决的问题尽量采用动态注册。
- 通过 registerReceive 注册的 BroadcastReceive 在对其进行注册的 Contex 对象“销毁”了或者调用了 unregisterReceive 方法后广播也就失效了；而通过<receiver>标签注册的 BroadcastReceiver 只要应用程序没有被删除就一直有效。

6.3.2 广播的分类

广播的分类标准不同，分类的结果也不同。如根据广播的接收顺序划分，有 Normal broadcasts(一般广播)和 Ordered broadcasts(有序广播)；根据发送广播的来源不同可以分为自定义广播和系统广播。本节主要探讨前一种划分。

一般广播是通过 Context.sendBroadcast 发出的，采用异步机制，匹配此广播的接收器都会响应，且响应的顺序不确定，无先后之分。这种方式对于利用广播而言，效率比较高，但各广播接收器无法改造广播信息，并且无法停止该广播。

有序广播是通过 Context.sendOrderedBroadcast 发出的，每次只能有一个接收器收到，与此广播匹配的接收器将按照先后顺序响应，先后顺序由广播接收器的优先级决定。每个接收器可以处理它所收到的广播数据，并将处理的数据传递给下一个接收器；也可以将所收到的广播扔掉，从而停止本次广播。广播被停止后，其他接收器将无法再响应。如果遇到优先级相同的广播接收器，仍按照随机方式响应。一般广播和有序广播接收广播的方式如图 6.8 所示。

有序广播的优先级取值范围在 −1000～1000，数值越大，优先级越高。使用 sendOrderedBroadcast 方法发送有序广播时，需要一个权限参数，如果为 null 则表示不要求接收者声明指定的权限，如果不为 null，则表示接收者若要接收此广播，需声明指定权限。这样做是从安全角度考虑的。例如，Android 系统的短信就是有序广播的形式，如果开放一个拦截垃圾短信功能的应用程序，当短信到来时可以先接收短信广播，必要时终止广播传递，这样的应用程序就必须声明接收短信的权限。

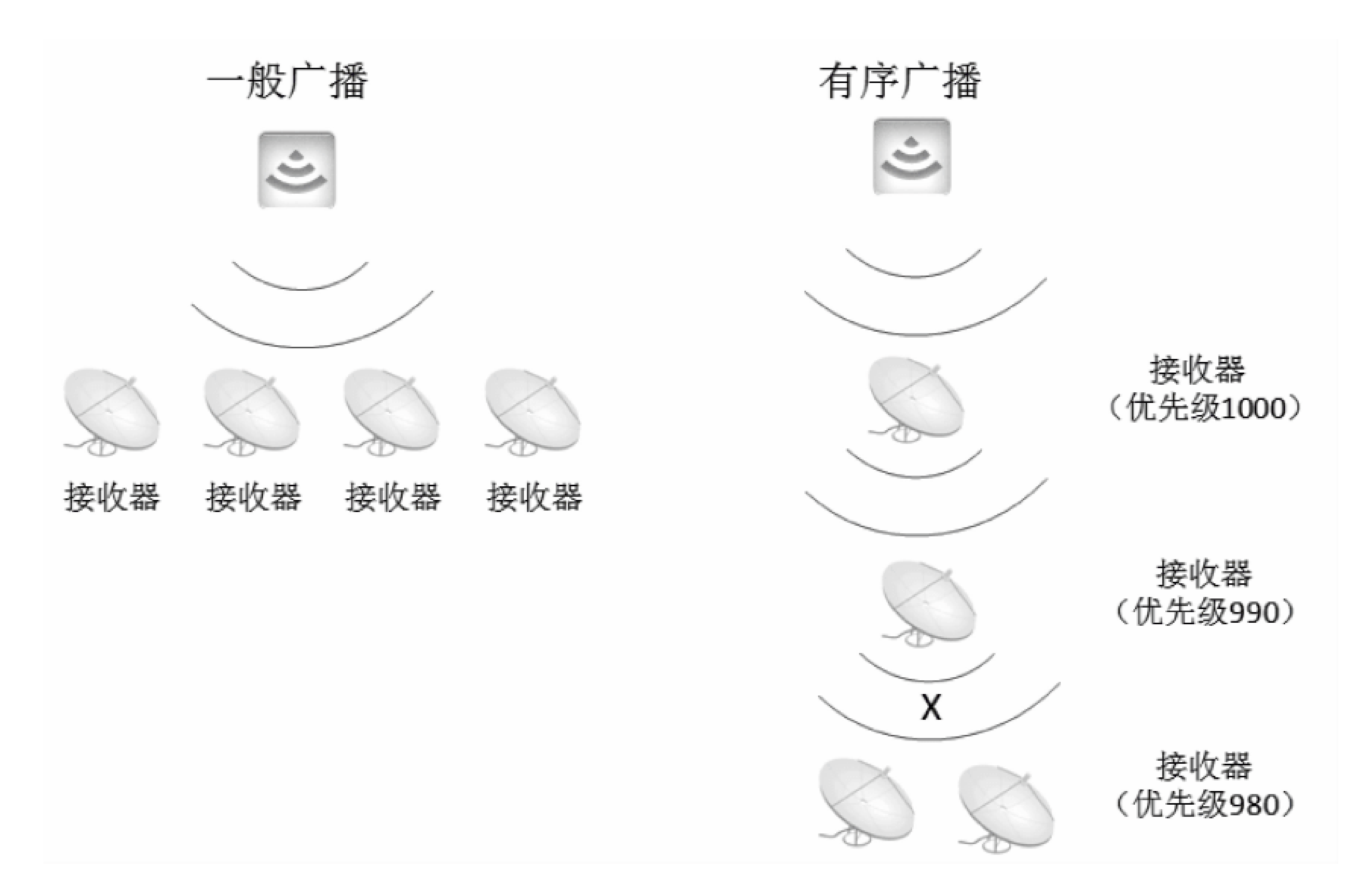


图 6.8 一般广播与有序广播

新建 part06_2 项目，完整代码参考随书配套资料。MainActivity 为主 Activity，布局文件是 main.xml，内有一个按钮，用于发出有序广播。新建 MyReceiver1、MyReceiver2 和 MyReceiver3，继承 BroadcastReceiver，代码如代码 06-15～代码 06-17 所示。

📖代码 06-15

```
public class MyReceiver1 extends BroadcastReceiver {
    static final String TAG = "Tag";
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取随 Intent 到达的数据
        String msg = intent.getStringExtra("msg");
        Log.i(TAG, "接收器 MyReceiver1,收到信息》" + msg);
        Bundle b  = new Bundle();
        b.putString("msg", msg + "@ from MyReceiver1");
        setResultExtras(b);
    }
}
```

MyReceiver1 接收器获取广播中的数据，输出到日志。创建 Bundle，用于存放数据。将从广播中获取的数据修改，在原来内容的基础上连接@ from MyReceiver1，并将新数据放入结果集。

📖代码 06-16

```
public class MyReceiver2 extends BroadcastReceiver {
    static final String TAG = "Tag";
    @Override
    public void onReceive(Context context, Intent intent) {
```

```
            String msg = getResultExtras(true).getString("msg");
            Log.i(TAG, "接收器 MyReceiver2,收到信息 »" + msg);
            Bundle b  = new Bundle();
            b.putString("msg", msg + "@ from MyReceiver2");
            setResultExtras(b);
            abortBroadcast();     //终止广播
        }
    }
```

请注意，MyReceiver2 接收器在提取数据时，是从结果集中获取的，并输出到日志。然后重建 Bundle，将获取的数据再次修改，追加@ from MyReceiver2，放入结果集。调用 abortBroadcast 方法（这是. BroadcastReceiver 中声明的 final 方法），终止广播。

代码 06-17

```
public class MyReceiver3 extends BroadcastReceiver {
    static final String TAG = "Tag";
    @Override
    public void onReceive(Context context, Intent intent) {
        String msg = getResultExtras(true).getString("msg");
        Log.i(TAG, "接收器,收到信息 »" + msg);
    }
}
```

MyReceiver3 接收器获取结果集中的数据，并将数据输出到日志。

修改 MainActivity 中的代码，参考代码 06-18，完整代码请参考随书配套资料 part06_2 项目。在 onStart 方法中注册三个广播接收器，在按钮监听器中发出有序广播。sendOrderedBroadcast 方法中两个参数的含义如下。

（1）intent：打算发出的广播，与此广播匹配的所有接收器将会响应。

（2）receiverPermission：权限字符串，可以引用系统值，也可以是自己定义的，一旦设定权限，则接收器必须声明该权限才能收到广播；如果为 null，则接收广播不需要权限（代码 06-18 中便是如此）。

代码 06-18

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    b = (Button) findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View v) {
            Intent intent  = new Intent("com.freshen.code.MyReceiver");
            intent.putExtra("msg", "原始广播数据");
            sendOrderedBroadcast(intent, null);     //发出有序广播
            Log.i("Tag", "有序广播已发出");
        }}
```

```
        );
    }
    @Override
    protected void onStart() {
        super.onStart();
        //注册监听器
        br1 = new MyReceiver1();
        IntentFilter if1 = new IntentFilter("com.freshen.code.MyReceiver");
        if1.setPriority(1000);              //设定权限
        registerReceiver(br1, if1);
        //
        br2 = new MyReceiver2();
        if1.setPriority(999);
        registerReceiver(br2, if1);
        //
        br3 = new MyReceiver3();
        if1.setPriority(998);
        registerReceiver(br3, if1);
    }
```

动态注册广播接收器，MyReceiver1 的权限为 1000，依次减小。根据有序广播的特性，会从优先级高的接收器依次传递给优先级低的接收器。代码 06-18 的广播应该按照 MyReceiver1->MyReceiver2->MyReceiver3 的顺序传递。但由于在 MyReceiver2 中执行了 abortBroadcast，本次广播在 MyReceiver2 后就中断了。代码运行效果如图 6.9 所示。

Level	Time	PID	Application	Tag	Text
I	0...	649	com.freshen.code	Tag	有序广播已发出
I	0...	649	com.freshen.code	Tag	接收器MyReceiver1，收到信息 》 原始广播数据
I	0...	649	com.freshen.code	Tag	接收器MyReceiver2，收到信息 》 原始广播数据@ from MyReceiver1

图 6.9 有序广播接收记录

有序广播中常有的方法解释如表 6.3 所示。Android 系统中的来电、来短信都会发出有序广播，只要声明优先级较高的接收器就可以拦截到这些广播，并决定是否终止该次广播，当然拦截系统广播需要声明权限，这是 6.3.3 节将要介绍的内容。

表 6.3 BroadcastReceiver 中的常用方法介绍

方 法 名	作 用
abortBroadcast()	终止广播，只能终止有序广播，对于一般广播无效
getResultExtras(boolean makeMap)	返回值为 Bundle，获取结果集数据
setResultExtras(Bundle extras)	设置结果集
isOrderedBroadcast()	判断当前广播是否是有序广播
onReceive (Context context, Intent intent)	抽象方法，实现广播接收器，必须实现该方法，接收到广播时执行该方法

6.3.3 权限与系统广播

BroadcastReceiver 的设计初衷是从全局考虑的，可以方便应用程序和系统、应用程序之间、应用程序内的通信，所以对单个应用程序而言 BroadcastReceiver 是存在安全性问题的。对此可以采用权限或声明是否接收外部广播来解决。此处主要讨论权限问题。

当应用程序发出某个广播时，系统会将发送的 Intent 与系统中所有注册的 BroadcastReceiver 的 IntentFilter 进行匹配，若匹配成功则执行相应的 onReceive 方法。如果在发出广播时，使用 sendBroadcast(Intent, String)方法，则该广播就具有了权限，发送给接收者必须具备相应的权限(permission)。

新建 part06_3 项目，完整代码请参考随书配套资料。MainActivity 作为主 Activity 存在，布局文件中有一个按钮用于发出通知。发出通知时的代码如代码 06-19 所示。sendBroadcast 方法的第二个参数为权限，该权限是自定义的，声明部分在 Manifest 文件中。

代码 06-19

```
Intent intent = new Intent("com.freshen.code.Receiver");
intent.putExtra("msg", "拥有权限才能收到本广播.");
sendBroadcast(intent, "com.freshen.code.Receiver.permission");
//sendBroadcast(intent);
Log.i("Tag", "广播已发出.");
```

打开 Manifest 文件，在<application>标签前面添加代码 06-20。<permission>标签用于声明自定义权限，android:name 是权限名，android:protectionLevel 是权限的风险级别，取值与功能描述参考表 6.4。赋予权限采用标签<uses-permission>。项目运行结果如图 6.10 所示。读者可以将权限去掉，再运行一遍，查看输出信息。也可以在 part06_2 项目中向本项目发出广播，验证没有授权的情况下能否接收到广播。

表 6.4 权限风险取值

取值	意义
normal	权限是低风险的，不会对系统、用户或其他应用程序造成危害
dangerous	权限是高风险的，系统将可能要求用户输入相关信息，才会授予此权限
signature	只有当应用程序所用数字签名与声明此权限的应用程序所有数字签名相同时，才能将权限授给它
signatureOrSystem	权限授给具有相同数字签名的应用程序或 Android 包类

代码 06-20

```
<!-- 声明权限 -->
<permission android:name="com.freshen.code.Receiver.permission"
    android:protectionLevel="normal"></permission>
<!-- 声明拥有的权限 -->
<uses-permission android:name="com.freshen.code.Receiver.permission"/>
```

Time	PID	Application	Tag	Text
0...	831	com.freshen.code	Tag	广播已发出。
0...	831	com.freshen.code	Tag	From Receiver3 拥有权限才能收到本广播。
0...	831	com.freshen.code	Tag	From Receiver1 拥有权限才能收到本广播。
0...	831	com.freshen.code	Tag	From Receiver2 拥有权限才能收到本广播。

图 6.10 具有权限的广播记录

Android 系统中常用的系统广播参考表 6.5，在接收这些广播时需要开启相应权限。

表 6.5 部分系统广播

广 播	含 义
android.provider.Telephony.SMS_RECEIVED	接收到短信时的广播
android.intent.action.NEW_OUTGOING_CALL	接收来电广播
android.intent.action.BATTERY_CHANGED	接收电池电量变化广播
android.intent.action.BOOT_COMPLETED	接收开机广播
android.net.conn.CONNECTIVITY_CHANGE	接收网络状态发生变化时的广播
android.intent.action.AIRPLANE_MODE	接收飞行模式广播
android.intent.action.CAMERA_BUTTON	接收打开照相机广播
android.intent.action.DEVICE_STORAGE_LOW	存储空间不足广播
android.intent.action.SCREEN_OFF	屏幕关闭广播
android.intent.action.SCREEN_ON	屏幕打开广播
android.intent.action.PACKAGE_FULLY_REMOVED	移除应用程序广播

新建项目 part06_4，完整代码请参考随书配套资料。创建广播接收器 SmsReceiver，如代码 06-21 所示。当短消息到达时会发出广播，该接收器实现响应该广播，并接收短信内容的功能。

📖 代码 06-21

```
@Override
    public void onReceive(Context context, Intent intent) {
        Log.i("Tag", "有短信");
        //
        Object [] pdus = (Object[]) intent.getExtras().get("pdus");
        for(Object o:pdus){
            byte [] pdu = (byte[]) o;
            //创建短信
            SmsMessage sms = SmsMessage.createFromPdu(pdu);
            //短信内容
            String content = sms.getMessageBody();
            //来电时间
            Date date = new Date(sms.getTimestampMillis());
            SimpleDateFormat sdf  = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
            String fdate = sdf.format(date);
            //来电号码
            String from  = sms.getOriginatingAddress();
            Log.i("Tag",content + " " + fdate + " [from] " + from);
        }
    }
```

短信发出广播时，内容会存储在 Bundle 对象中以 pdus 为键的对象数组。遍历该数组，将每条短消息转换为字节数组。调用 SmsMessage（短信类）类的静态方法创建短信对象。对于短信类的使用，可以查阅 API，了解上述代码中获取短信内容、发送时间和来电号码的操作。

如果要实现拦截短信功能，检测短信内容后，经该广播终止，用户就无法收到该条短信。该接收器需要较高的优先级。

响应短信广播需要开启权限<uses-permission android:name="android.permission.RECEIVE_SMS"/>，该短信接收器的配置信息如代码 06-22 所示。

代码 06-22

```
<receiver android:name = ".SmsReceiver">
    <intent - filter>
        <action android:name = "android.provider.Telephony.SMS_RECEIVED" />
        <category android:name = "android.intent.category.DEFAULT" />
    </intent - filter>
</receiver>
```

在虚拟机中测试收发短信需要切换到 DDMS 视图模式。在 Eclipse 中选择 Window→Open Perspective→DDMS 命令。在 Emulator Control 窗口，Telephone Actions 面板中可以模拟电话和短信功能，如图 6.11 所示。Incoming number 为来电号码，这个可以随意输入；Voice 为来电呼叫；SMS 为发送短信，内容是 Message 输入框中的文本。设定完成后单击 Send 按钮就可以进行呼叫了。

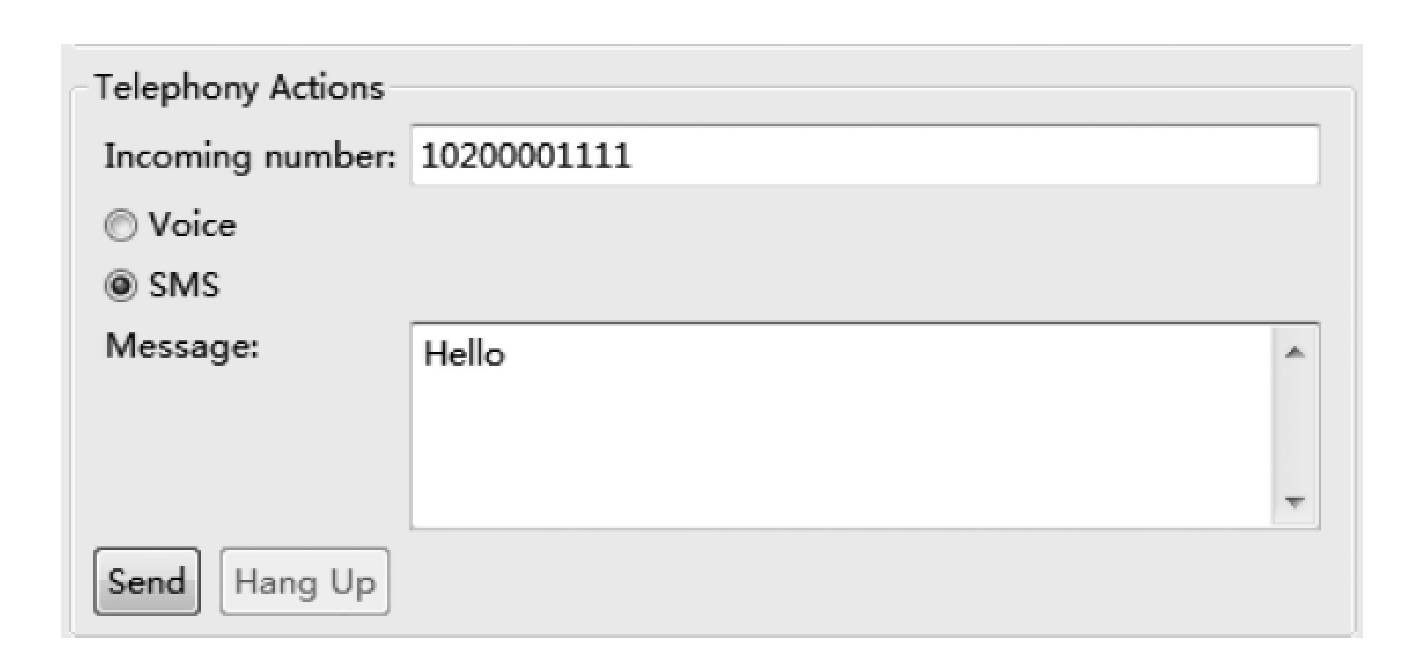


图 6.11 虚拟机中模拟发送短信

项目 part06_4 的执行结果如图 6.12 所示。

Level	Time	PID	Application	Tag	Text
I	0...	1387	com.freshen.code	Tag	有短信
I	0...	1387	com.freshen.code	Tag	Hello 2013-05-25 11:16:41 [from] 10200001111

图 6.12 响应并接收短信广播

6.4 ContentProvider

Android 系统将所有数据都规定为对外私有，也就是无法直接访问应用程序之外的数据。如果需要访问其他程序的数据或向其他程序提供数据，就需要使用 ContentProvider（数据供应器）。该类位于 android.content 包中，为存储和获取数据提供了统一的接口。ContentProvider 对数据进行了封装，在使用时不用关心数据存储的细节。

6.4.1 使用 ContentProvider

Android 为常见的一些数据提供了默认的 ContentProvider，包括音频、视频、图片和通讯录等，使用时需要得到相应访问的权限。ContentProvider 为存储和读取数据提供了统一的接口，经常涉及的方法如表 6.6 所示。

表 6.6 ContentProvider 常用方法说明

返回值	方法	说明
Cursor	query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	查询操作，通过 uri 查询数据，返回 Cursor（这个查询语法类似于 Hibernate）
Uri	insert(Uri uri, ContentValues values)	插入操作，将数据插入到 uri 指定内容，数据需要封装到 ContentValues 对象中
int	update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	更新操作
int	delete(Uri uri, String selection, String[] selectionArgs)	删除操作

外界程序在访问 ContentProvider 中的数据时，并不是进行直接操作，而是借助 ContentResolver 类，该类也位于 android.content 包中，多数方法都是由 static 或 final 修饰，它是作为一个工具类来使用的。ContentResolver 对象可以通过 getContentResolver() 方法（Context 类的方法）得到，其中提供的方法与 ContentProvider 类中的方法对应。

使用 ContentProvider 对象读取数据涉及的类有 Uri（数据统一标识）、ContentResolver（读取数据的功能）、Cursor（包含结果数据的游标）。下面通过一个读取联系人数据的程序演示 ContentProvider 的使用。该演示代码使用系统 provider，系统提供的 provider 位于 android.provider 包。ContactsContract 是处理联系人的类，该类的使用比较复杂，在此不展开介绍此类的使用。该部分代码对应项目 part06_5，需要处理的代码有三处：MainActivity.java 查看联系人的主界面；main.xml 布局文件；Manifest 配置文件，需要开启权限<uses-permission android:name="android.permission.READ_CONTACTS"/>，允许读取联系人信息。代码 06-23 演示读取联系人信息的核心功能，运行效果如图 6.13 所示。（运行前需要在虚拟机中添加联系人。）

📖 **代码 06-23**

```
//查询所有联系人信息
public void showAllContacts(){
    //构造需要查询信息的字段
    String data[] = {ContactsContract.Data._ID,              //联系人 ID 唯一、必需
            ContactsContract.Data.DISPLAY_NAME,           //联系人姓名
            ContactsContract.Data.MIMETYPE,
            ContactsContract.Data.DATA1};                 //电话
    //查询指定资源的数据
    Cursor cursor = managedQuery(ContactsContract.Data.CONTENT_URI, data, null, null, null);
    cursor.moveToFirst();
    StringBuffer txt = new StringBuffer();
    while(!cursor.isAfterLast()){
        Log.i("Tag", cursor.getString(2));
        //只显示有电话号码的联系人
        if(cursor.getString(2).equalsIgnoreCase(Phone.CONTENT_ITEM_TYPE)){
            txt.append("ID: " + cursor.getString(0) + "\t");
            txt.append("Name: " + cursor.getString(1) + "\t");
            txt.append("Phone: " + cursor.getString(3));
            txt.append('\n');
        }
        cursor.moveToNext();
    }
    tv.setText(txt);
}
```

代码 06-23 最核心的方法是 managedQuery()，该方法是 Activity 中的一个 final 方法，用于查询指定资源的数据，该方法现已不鼓励使用了。结合前面的介绍，此处使用代码 06-24 演示的方式查询，效果一样。



图 6.13 读取系统联系人

📖 **代码 06-24**

```
ContentResolver cr = getContentResolver();
Cursor cursor = cr.query(ContactsContract.Data.CONTENT_URI, data, null, null, null);
```

ContentResolver 类的 query 方法与 managedQuery 方法具有相同的参数要求，具体说明如表 6.7 所示。关于对数据增、删、改、查操作中参数的说明详见数据保存章节。

表 6.7 查询方法参数明细

参数名	作　用
Uri uri	待查询数据(Content Provider)的 Uri
String[] projection	需要查询的字段，相当于 select 语句后面的字段信息
String selection	查询的条件，相当于 where 子句后面的条件
String[] selectionArgs	查询条件的值，给 where 子句中的占位参数传值
String sortOrder	排序条件，相当于 Order by 子句

ContactsContract.Data.CONTENT_URI 是联系人数据的 Uri，是系统级 Uri。data 数组保存需要查询的字段值，都引用了 ContactsContract.Data 类的属性。ContactsContract.Data.MIMETYPE 属性是标明该数据是何种类型的信息。（对于联系人来说，电话号码、邮件地址、住址等都是信息，这些信息是需要区分的。）代码 06-23 中只选择显示符合 Phone.CONTENT_ITEM_TYPE 类型的信息，即为电话号码。

查询的结果会返回 Cursor 对象，即游标，可以移动，具有生命周期，在数据保存章节再详细介绍 Cursor 的使用，此处先按照代码 06-23 演示的方式获取 Cursor 中的数据即可。

总体而言，使用 ContentProvider 查询其他程序中主动暴露出的数据比较简单。首先，确定需要查询数据的 Uri（Android 系统中有很多原生态的 Uri）；其次，使用 ContentResolver 类中的 query 方法查询；最后，检索 Cursor，处理相应数据。

6.4.2 Uri

Uri 类位于 android.net 包中，代表了要操作的数据，主要包含两部分信息：需要操作的 ContentProvider 与 ContentProvider 中的具体数据。一个 Uri 由以下几部分组成，见图 6.14。（此处以 ContentProvider 中的 Uri 为例。）

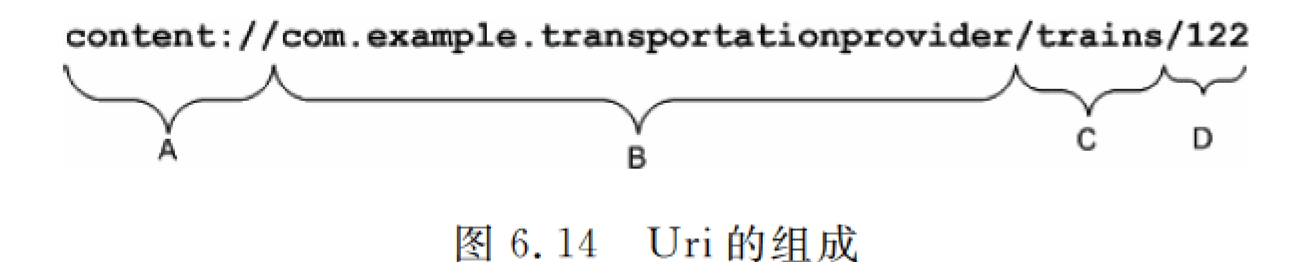


图 6.14 Uri 的组成

（1）A：前缀标识，任何 Uri 都有一个固定的前缀，content://标明这是 ContentProvider 中使用的 Uri。

（2）B：数据的权限标识，要求唯一。

（3）C：路径标识，代表 Uri 所要访问的具体数据表。

（4）D：ID 信息，C 中数据表的某一个 ID 值。

在编程中一般不会直接使用 Uri 来标识某个 ContentProvider，而是选择使用对应的常量。如代码 06-23 中所示，联系人数据表的 Uri 是 ContactsContract.Data.CONTENT_URI 常量，它对应的具体 Uri 字符串是 content://com.android.contacts/data。这个 Uri 只有三个组成部分，说明需要访问 data 表中的全部数据，如果改为 content://com.android.contacts/data/2，则表明访问 data 表中 ID 为 2 的那条记录。

将合法的 Uri 字符串转变为 Uri 对象，可以使用 Uri.parse( ) 方法，如 ContactsContract.Data.CONTENT_URI 对象可以通过 Uri.parse(content://com.android.contacts/data)获得。由于版本更新过程中可能出现 Uri 字符串不一致的现象，所有建议多采用系统常量。

Uri 代表了要操作的数据，经常需要解析 Uri，并从 Uri 中获取数据。Android 系统提供了两个用于操作 Uri 的工具类，分别为 UriMatcher 和 ContentUris，都位于 android.content 包中。

UriMatcher 类用于匹配 Uri 的检验。主要涉及方法 addURI(String authority，String path，int code)和 match(Uri uri)。代码 06-25 演示了 UriMatcher 类的使用。

📖 **代码 06-25**

```
//匹配检验
UriMatcher uMatcher = new UriMatcher(UriMatcher.NO_MATCH);
uMatcher.addURI("com.android.contacts", "data", 1);
int r = uMatcher.match(ContactsContract.Data.CONTENT_URI);
Log.i("Tag", "匹配结果:" + r);
```

创建 UriMatcher 对象，传入常量 NO_MATCH，当匹配不成功时会返回该常量的值。使用 addURI 方法添加匹配样式，这相当于模板，供待检验的 Uri 与此比对，可以多次调用此方法，添加多个模板。该方法的三个参数，第一个是 Uri 的数据权限；第二个是路径标识，可以使用通配符（* 用于通配文本，# 用于通配数字）；第三个参数是匹配成功时的返回值，通过与此值比对，可以判定是否匹配成功。match 方法用于匹配 Uri，若匹配成功，返回 addURI 方法中的第三个参数，若不成功，返回构造 UriMatcher 时传入的值。

ContentUris 类用于操作 Uri 路径后面的 ID 部分，它有如下两个比较实用的方法。

（1）static Uri withAppendedId(Uri contentUri, long id)，向 Uri 路径后面追加 id。

（2）static long parseId(Uri contentUri)，获取 Uri 中的 id 值。

6.4.3 ContentProvider 基本操作

ContentProvider 有 4 种基本操作：查询（query）、插入（insert）、更新（update）、删除（delete），都可以通过其助手 ContentResolver 实现。请一定记住 ContentResolver 对象可以通过 Context 对象的 getContentResolver()方法获取。

新建项目 part06_6，实现对联系人的增、删、改、查 4 种操作。MainActivity.java 为主 Activity，布局文件为 main.xml，ListView 控件的布局文件是自定义的 listview_item.xml，对联系人进行读写操作需要开启以下两个权限。

（1）读权限：<uses-permission android:name="android.permission.READ_CONTACTS"/>。

（2）写权限：<uses-permission android:name="android.permission.WRITE_CONTACTS"/>，可操作插入、更新和删除操作。

完整代码请查阅随书配套资料，下面依次介绍实现 4 种操作的方法。当单击“查询”按钮，执行 queryData 方法。

📖 **代码 06-26**

```
public void queryData(){
    String fields[] = {ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME,
            ContactsContract.Contacts.Data.DATA1
        };
    String q_id = et_queryId.getText().toString();
    Log.i("Tag", "id = " + q_id);
    int n = 0;
    if(q_id!= null&&q_id.length()<3){
        try {
```

```
                n = Integer.parseInt(q_id);
                cursor = cr.query(contactsUri, fields, "_id = ?",new String[ ]{q_id}, null);
            } catch (NumberFormatException e) {
                Toast.makeText(this, "ID 错误或未指定!", Toast.LENGTH_SHORT).show();
                n = 0;
            }
        }
        if(n == 0){
            cursor = cr.query(contactsUri, fields,null,null, null);
            Log.i("Tag", "查询所有联系人 !");
        }
        Log.i("Tag", "共有联系人: " + cursor.getCount());
        //填充 ListView
        if(cursor!= null){
            SimpleCursorAdapter sca = new SimpleCursorAdapter(this,R.layout.listview_item,
                    cursor,fields,new int[ ]{R.id.listview_item_id,R.id.listview_item_name,
                                        R.id.listview_item_phone});
            lv.setAdapter(sca);
        }
    }
```

fields 数组是欲查询字段信息,其值都是引用联系人对象 Contacts 的属性。q_id 变量是当用户指定 ID 查询时,收入的 ID 值。if 判定是否指定 ID 查询,确定执行 cr.query (contactsUri, fields, "_id=?",new String[]{q_id}, null)查询或 cr.query(contactsUri, fields,null,null, null)查询,cr 即为 ContentResolver 的引用。当输入 ID 进行条件查询时,需要将条件传入,参数的意义请参考表 6.7。查询的结果会返回 cursor 对象,直接构造 SimpleCursorAdapter 数据适配器,设置给 ListView(适配器的用法可以查询前面章节),即可显示查询结果。

单击“插入”按钮,执行 insertData 方法。联系人插入操作涉及联系人 ContactsContract 的使用,Android 系统将联系人定义为多张表,联系人的不同信息都存放在不同表中,并通过_ID 作为约束。插入联系人基本信息可以按照代码 06-27 演示的方法进行。

代码 06-27

```
public void insertData(){
    String name = et_insertName.getText().toString();
    String phone = et_insertPhone.getText().toString();
    if(name == null||name.length()<0)return;
    ContentValues cv = new ContentValues();
    //插入新的联系人,需要 ContactID,采用下面的方法获取
    Uri rawUri = cr.insert(RawContacts.CONTENT_URI, cv);
    long contactID = ContentUris.parseId(rawUri);            //获取 ID
    //插入联系人要分步完成
    //step 1 插入姓名
    cv.clear();
    cv.put(Data.RAW_CONTACT_ID, contactID);
    cv.put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);
    cv.put(StructuredName.GIVEN_NAME, name);                 //联系人姓名
    cr.insert(Data.CONTENT_URI, cv);                         //执行插入
    Log.i("Tag", "插入姓名完成");
```

```
        //step 2 插入号码
        cv.clear();
        cv.put(Data.RAW_CONTACT_ID, contactID);
        cv.put(Data.MIMETYPE , Phone.CONTENT_ITEM_TYPE);
        cv.put(Phone.NUMBER, phone); //电话号码
        cv.put(Phone.TYPE, Phone.TYPE_MOBILE); //号码类型,移动电话
        cr.insert(Data.CONTENT_URI, cv); //执行插入
        Log.i("Tag", "插入号码完成");
        //如果后面还有邮箱等其他信息,则需要另做插入操作
    }
```

新增联系人信息时，需要先得到系统 ID，可以通过向 ContactsContract.RawContacts.CONTENT_URI 插入空值来获取，插入操作会返回能代表该插入内容的一个 URI，然后使用 ContentUris.parseId()方法，获取返回 URI 中的 ID 值。联系人的信息，如姓名、电话号码(可以设定类型)、邮件、地址等，都可以插入到 ContactsContract.Data.CONTENT_URI。

单击“更新”按钮执行 updateData 方法。cr.update 方法中参数的意义可以参照表 6.7 对 query 方法的介绍。

代码 06-28

```
    public void updateData(){
        String updateID = et_updateId.getText().toString();
        String updatePhone = et_updatePhone.getText().toString();
        if(updateID == null||updateID.length()< 1)return;
        ContentValues cv  = new ContentValues();
        cv.put(Phone.NUMBER,updatePhone);
        int n = cr.update(Data.CONTENT_URI, cv, "_ID = ?", new String[]{updateID});
        Log.i("Tag", "更新记录条数 " + n);
    }
```

单击“删除”按钮，执行 delData 方法。cr.delete 方法中参数的意义可以参考表 6.7 的介绍。该项目的运行效果如图 6.15 所示。



图 6.15 联系人 SIUD 操作

📖 **代码 06-29**

```
public void delData(){
    String delID = et_delId.getText().toString();
    if(delID == null|| delID.length()< 1)return;
    int n = cr.delete(Data.CONTENT_URI, "_ID = ?", new String[]{delID});
    Log.i("Tag", "删除记录 条数: " + n);
}
```

6.5 Intent 与 IntentFilter

Intent 对象位于 android.content 包中，虽没有位列四大组件，但它的作用相当重要，它既可以用来启动 Activity、Service、BroadcastReceiver，也可以用于传输数据。Intent 对象负责对应用程序中一次操作的动作、属性、动作涉及数据、附加数据等进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。

Intent 对象包含 Component、Action、Category、Data、Type、Extra 和 Flag 属性。其中，Component 是指明需要启动的目标组件；Action 是指 Intent 要完成的一个动作；Category 是对 Action 附加的信息；Data 是向 Action 属性提供操作的数据，具体是某个数据对应的 Uri；Type 属性是指明 Data 的 MIME 类型；Extra 属性常用于传输数据；Flag 属性用于为 Intent 增加额外说明。

6.5.1 Component、Action 与 Category

Intent 在启动另一个组件时有两种方式，一种是可以明确指定组件类型名，此时需要设置 Component 属性(这种 Intent 称为显式 Intent)；另一种是比较宽泛的指明打算启动组件应该满足的特征，这种不需要设置 Component 属性，一旦 Android 发现某个组件满足要求，便直接启动(这种 Intent 称为隐式 Intent)。

显式 Intent 设置 Component 采用以下方法：

public Intent setComponent (ComponentName component)

参数是 ComponentName 类型，该类位于 android.content 包中，可以通过 new 直接创建对象。比如从 ActivityA 跳转到 ActivityB，该对象可以构建成 new ComponentName (Activity.this，ActivityB.class)。这种写法看起来非常眼熟，因为 Intent 类直接提供了创建显式 Intent 对象的构造方法，所以上述 Activity 跳转可以写成 new Intent(Activity.this，ActivityB.class)，这也是前面介绍 Intent 时采用的方法。对于创建显式 Intent，启动指定组件的代码此处不再给出。

下面演示隐式 Intent 的用法。项目 part06_7 中 MainActivity 是主界面，布局文件是 activity_main.xml，只有一个按钮，通过单击按钮跳转到 ShowActivity(布局文件是 showlayout.xml)。MainActivity 中的核心代码如代码 06-30 所示，其他文件请查看随书配套资料。

📖 **代码 06-30**

```
public final static String SHOW_ACTION = "com.freshen.intent.action.SHOWACTIVITY";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findViewById(R.id.button1).setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View arg0) {
            //创建 Intent,没有指明 Component 属性
            Intent in = new Intent();
            //设定 Action
            in.setAction(SHOW_ACTION);
            //开启其他 Activity
            startActivity(in);
        }
    });
}
```

代码 06-30 中没有明确指明所启动的组件，只是给 Intent 对象设置了 action 属性。那么 startActivity()方法究竟启动哪个 Activity 呢？查看配置文件，ShowActivity 的配置信息如代码 06-31 所示。相比以前的 Activity 配置，此处多了<intent-filter>标签，该标签的作用在于声明它所在 Activity 可以响应的 Intent。<intent-filter>标签内部可以配置<action>、<category>、<data>子标签，<action>标签具体指明了 Activity 可以响应的 Intent。对比代码 06-30 和代码 06-31 中的粗体字，可以发现它们的匹配关系。除此之外，<intent-filter>标签内部还需要配置一个<cagegory>标签，这是因为 Intent 对象创建时会默认 Category 属性 android.intent.category.DEFAULT。

📖 **代码 06-31**

```
<activity
    android:name = "com.freshen.code.ShowActivity"
    android:label = "@string/title_activity_show" >
    <intent - filter>
        <action android:name = "com.freshen.intent.action.SHOWACTIVITY" />
        <category android:name = "android.intent.category.DEFAULT" />
    </intent - filter >
</activity>
```

字符串 com.freshen.intent.action.SHOWACTIVITY 是自己定义的，只要确保它的唯一性即可。Android 系统中有很多这样的 action 字符串，用于启动系统 Activity。表 6.8 列举了部分 action 属性，这些都是系统中已有的 action，可以用于启动系统中的应用。

表 6.8 Standard Activity Actions(部分)

常量名	字 符 串 值	说 明
ACTION_MAIN	android.intent.action.MAIN	表示程序的入口
ACTION_VIEW	android.intent.action.VIEW	向用户显示数据

续表

常量名	字 符 串 值	说　　明
ACTION_EDIT	android. intent. action. EDIT	请求一个 Activity 编辑 URI 处的数据
ACTION_PICK	android. intent. action. PICK	启动一个子 Activity 从 URI 挑选一个项目，当关闭时，返回指向被挑选项目的 URI
ACTION_CHOOSER	android. intent. action. CHOOSER	弹出 Activity 选择器，让用户选择
ACTION_GET_CONTENT	android. intent. action. GET _CONTENT	让用户选择数据，并返回所选数据
ACTION_DIAL	android. intent. action. DIAL	启动一个电话拨号程序，使用预置在数据 URI 中的号码来拨号
ACTION_CALL	android. intent. action. CALL	启动电话拨号工具，并立即用数据 URI 中的号码初始化一个呼叫
ACTION_SEND	android. intent. action. SEND	启动一个 Activity 来发送特定的数据
ACTION_SENDTO	android. intent. action. SENDTO	启动一个 Activity 来给 URI 中的指定联系人发送一个消息
ACTION_ANSWER	android. intent. action. ANSWER	打开一个 Activity 来处理来电

表 6.9 列举了部分标准 category 属性，这些属性可以通过 addCategory(String)方法添加给 Intent 对象。

表 6.9　Standard Categories(部分)

Category	常量对应字符串	说　　明
CATEGORY_DEFAULT	android. intent. category. DEFAULT	默认的 Category
CATEGORY_BROWSABLE	android. intent. category. BROWSABLE	指定该 Activity 能被浏览器安全调用
CATEGORY_TAB	android. intent. category. TAB	指定 Activity 作为 TabActivity 的 Tab 页
CATEGORY_LAUNCHER	android. intent. category. LAUNCHER	Activity 显示顶级程序列表中
CATEGORY_INFO	android. intent. category. INFO	用于提供包信息
CATEGORY_HOME	android. intent. category. HOME	设置该 Activity 随系统启动而运行
CATEGORY_PREFERENCE	android. intent. category. PREFERENCE	该 Activity 是参数面板
CATEGORY_TEST	android. intent. category. TEST	该 Activity 是一个测试
CATEGORY_CAR_DOCK	android. intent. category. CAR_DOCK	指定手机被插入汽车底座(硬件)时运行该 Activity
CATEGORY_DESK_DOCK	android. intent. category. DESK_DOCK	指定手机被插入桌面底座(硬件)时运行该 Activity
CATEGORY_CAR_MODE	android. intent. category. CAR_MODE	设置该 Activity 可在车载环境下使用

项目 part06_8 使用 ACTION_PICK 属性打开系统联系人，选择联系人，并将联系人信息带回 Activity。主 Activity 是 MainActivity. java，核心功能如代码 06-32 所示，布局文件

是 activity_main. xml。

代码 06-32

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.textView1);
        bt = (Button) findViewById(R.id.button1);
        bt.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();                                    ┐
                intent.setAction(Intent.ACTION_PICK);                            │
                intent.setData(ContactsContract.Contacts.CONTENT_URI);           ├ ①
                startActivityForResult(intent,1 );                               ┘
            }
        });
    }
    //当 Intent 开启的 Activity 返回时,执行
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        Log.i("Tag","Result " + requestCode + " , " + resultCode);
        //处理返回的数据
        if(requestCode == 1){
            Uri uri = data.getData();
            Log.i("Tag", "uri = " + uri);
            showContractInfo(uri);
        }
    }
    //显示联系人信息
    public void showContractInfo(Uri uri){
        //构造需要查询信息的字段
        String data[] = {ContactsContract.Data._ID,          //联系人 ID 唯一、必需
                ContactsContract.Data.DISPLAY_NAME,          //联系人姓名
                };                                           //电话
        //查询指定资源的数据
        Cursor cursor = managedQuery(uri, data, null, null, null);
        cursor.moveToFirst();
        StringBuffer txt = new StringBuffer();
        if (!cursor.isAfterLast()) {
            // 只显示有电话号码的联系人
            txt.append("ID: " + cursor.getString(0) + "\t");
            txt.append("Name: " + cursor.getString(1) + "\t");
            txt.append('\n');
            cursor.moveToNext();
        }
        tv.setText(txt);
    }
```

上述代码中在创建 intent 对象时，设置了系统 action 属性，并添加 data 属性，以实现启动一个子 Activity，从 URI 挑选一个项目，当关闭时，返回指向被挑选项目的 URI。为了得到子 Activity 中返回的数据，此处开启 Activity 的方法是 startActivityForResult(intent, 1)，其中第二个参数要大于 0，以便子 Activity 成功执行返回后，onActivityResult 方法得以执行(这很重要，若第二个参数小于 0，该方法等价于 startActivity)。

onActivityResult 方法的三个参数中，requestCode 是 startActivityForResult 方法执行时传入的第二个参数，resultCode 是由子 Activity 设置，data 保存子 Activity 返回的数据。

执行该项目，单击按钮会打开系统联系人 Activity，如图 6.16 所示，选择任意联系人，返回主 Activity，会显示带回的数据，如图 6.17 所示。



图 6.16　系统联系人

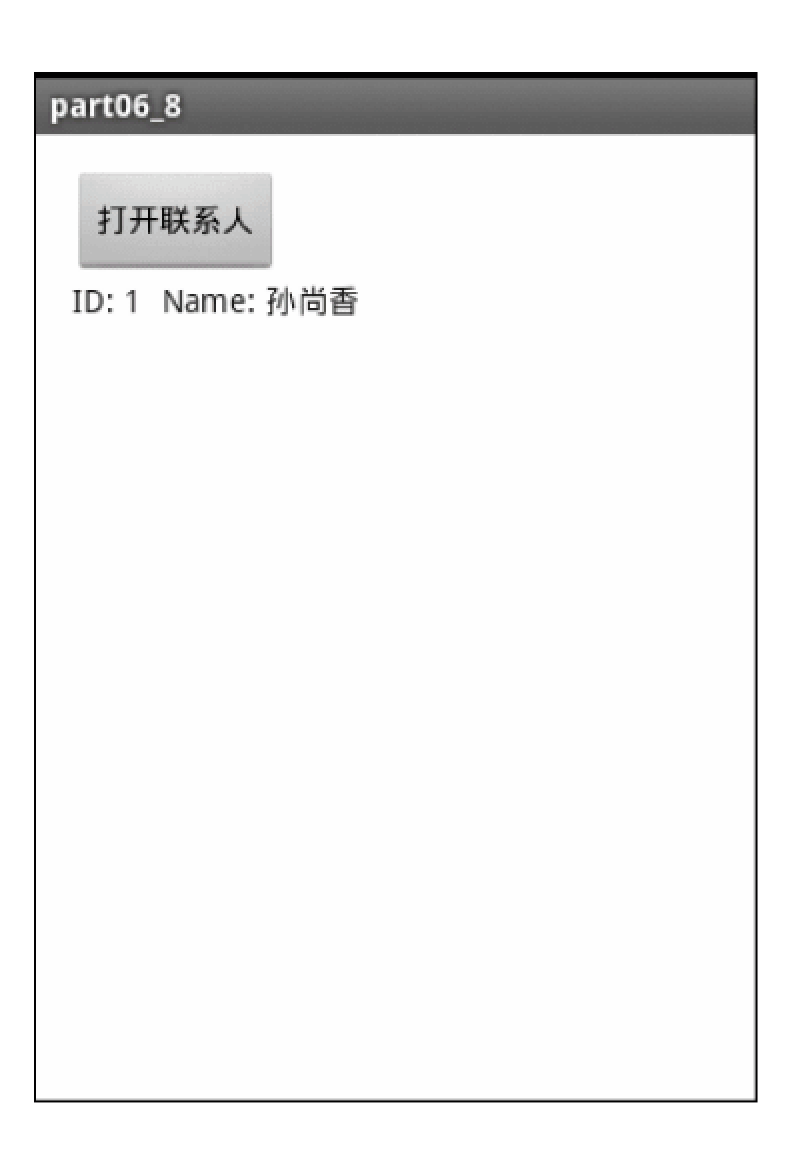


图 6.17　获取返回的数据

6.5.2　Data 与 Type 属性

Data 属性用于指明提供的数据，往往与 Type 属性一起使用。Intent 中配置这两个属性的方法如表 6.10 所示。

表 6.10　设置 Data 与 Type 的方法

返回值类型	方　法　名	说　　明
Intent	setData(Uri data)	设置 Data 属性，会覆盖 Type 属性
Intent	setType(String type)	设置 Type 属性，会覆盖 Data 属性
Intent	setDataAndType(Uri data, String type)	常用方法，可以同时设置 Data 和 Type 属性

在 Manifest 文件中配置 Data 和 Type 属性都是通过<data>标签，如代码 06-33 所示。

(1) android:mimeType，用于声明 Type 属性。

(2) android:scheme，用于声明 Data 属性 Uri 中 scheme 部分。

(3) android:host，用于声明 Data 属性 Uri 中 host 部分。

(4) android:path,用于声明 Data 属性 Uri 中 path 部分。

(5) android:pathPrefix,用于声明 path 部分的前缀。

(6) android:pathPattern,用于声明 path 的匹配表达式。

scheme,host,port,path,pathPrefix,pathPattern 是用来匹配 Intent 中的 Data Uri 的。具体规则如下(可以参考前面 URI 部分的介绍):

scheme://host:port/path or pathPrefix or pathPattern

📖 **代码 06-33**

```
<activity
    android:name = "com.freshen.code.MainActivity"
    android:launchMode = "singleTask"
    android:label = "@string/app_name" >
    < intent - filter>
        <action android:name = "android.intent.action.MAIN" />
        <category android:name = "android.intent.category.LAUNCHER" />
        <data
            android:mimeType = ""
            android:scheme = ""
            android:host = ""
            android:port = ""
            android:path = ""
            android:pathPrefix = ""
            android:pathPattern = "" />
    </intent - filter>
</activity>
```

6.5.3 Extra 与 Flag 属性

Extra 属性主要用于 Intent 携带数据传入另一个组件。查看 API 文档,Intent 类提供了大量的 put×××方法,用于传递不同类型的数据,其中 putExtras(Bundle extras)方法的参数是一个 Bundle(其实就是一个 Map),可以以键值对的形式将一组数据传递到另一个组件。

项目 part06_9 演示 Intent 携带数据在 Activity 直接跳转,其中发送数据的核心代码如代码 06-34 所示,接收数据如代码 06-35 所示。

📖 **代码 06-34**

```
/**
 * 登录按钮监听器
 * @param view
 * 触发 OnClick 事件的控件
 */
public void login(View view){
    Intent intent  = new Intent(MainActivity.this,LoginActivity.class);
    Bundle bu = new Bundle();
```

```
        bu.putString("loginName", et1.getText().toString());
        bu.putString("loginPwd", et2.getText().toString());
        intent.putExtras(bu);
        startActivity(intent);
    }
```

代码 06-35

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        //获取传递过来的 Intent
        Intent intent = getIntent();
        //获取 Bundle 对象
        Bundle bu = intent.getExtras();
        String ln = bu.getString("loginName");
        String lp = bu.getString("loginPwd");
        TextView tv = (TextView) findViewById(R.id.tv_login);
        tv.setText("账户: " + ln + "\n 密码: " + lp);
    }
```

Intent 类的 Flag 属性用于添加控制属性，使用 setFlags(int)和 addFlags(int)方法完成添加 Flag 属性。常用的 Flag 及其作用介绍如下。

(1) FLAG_ACTIVITY_BROUGHT_TO_FRONT：

这个标志一般不是由程序代码设置的，常在配置文件中，给 launchMode 属性设置 singleTask 模式时系统自动设定。

(2) FLAG_ACTIVITY_CLEAR_TOP：

如果 Activity 已经在当前的 Task 中运行，则不再重新构造这个 Activity 的实例，而是将这个 Activity 上方的所有 Activity 都关闭，原 Activity 开始运行。

(3) FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET：

如果设置，这将在 Task 的 Activity stack 中设置一个还原点，当 Task 恢复时，需要清理 Activity。

(4) FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS：

如果设置，新的 Activity 不会在最近启动的 Activity 的列表中保存。

(5) FLAG_ACTIVITY_FORWARD_RESULT

如果设置，并且这个 Intent 用于从一个存在的 Activity 启动一个新的 Activity，那么，这个作为答复目标的 Activity 将会传到这个新的 Activity 中。这种方式下，新的 Activity 可以调用 setResult(int)，并且这个结果值将发送给那个作为答复目标的 Activity。

(6) FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY：

这个标志一般不由应用程序代码设置，如果这个 Activity 是从历史记录里启动的(常按 Home 键)，那么，系统会帮助用户设定。

(7) FLAG_ACTIVITY_MULTIPLE_TASK：

不要使用这个标志，除非自己实现了应用程序启动器。

(8) FLAG_ACTIVITY_NEW_TASK:

如果设置,这个 Activity 会成为历史 stack 中一个新 Task 的开始。

(9) FLAG_ACTIVITY_NO_ANIMATION:

如果在 Intent 中设置,并传递给 Context.startActivity(),这个标志将阻止系统进入下一个 Activity 时应用 Acitivity 迁移动画。

(10) FLAG_ACTIVITY_NO_HISTORY:

如果设置,新的 Activity 将不在历史 stack 中保留。用户一离开它,这个 Activity 就关闭了。

(11) FLAG_ACTIVITY_REORDER_TO_FRONT:

如果在 Intent 中设置,并传递给 Context.startActivity(),这个标志将引发已经运行的 Activity 移动到历史 stack 的顶端。

(12) FLAG_ACTIVITY_SINGLE_TOP:

如果设置,当这个 Activity 位于历史 stack 的顶端运行时,不再启动一个新的。

给项目 part06_9 添加退出功能。在 LoginActivity 的布局文件 activity_login.xml 中添加一个按钮,当单击时执行 quitapp 方法,返回手机桌面,如代码 06-36 所示。

代码 06-36

```
/**
 * 当单击"退出"按钮时,退出应用程序,返回到手机桌面
 * @param view
 * 被单击的控件,此处是按钮
 * */
public void quitapp(View view){
    Intent intent = new Intent();
    intent.setAction(Intent.ACTION_MAIN);
    intent.addCategory(Intent.CATEGORY_HOME);
    //Activity 栈中目标 Activity 之上的 Activity 关闭
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(intent);
}
```

习　题

1. Service 的 onStart()方法何时会执行?
2. 与发送广播相关的方法有哪些?简要说明如何使用这些方法。
3. Uri 的组成部分有哪些?各代表什么意义?
4. ContentProvider 的基本操作有哪些?应该如何使用?
5. 如何使用隐式 Intent,请举例说明。



CHAPTER 7

第7章

2D游戏开发

本章主要内容：

游戏开发介绍；
View 与 SurfaceView；
Canvas 介绍；
Paint 介绍；
画布的操作；
绘制游戏元素；
屏幕坐标；
屏幕事件。

前面几章介绍了 Android 平台开发 APP 的基础知识，本章主要介绍在 Android 平台进行游戏开发的基础知识。在游戏开发中一般很少使用 Android 系统提供的组件（如按钮、输入框等），游戏元素（人物、物品、效果等）大都是通过画布和画笔绘制出来的。游戏开发需谨记游戏对于用户是动态的，游戏对于开发者是静态的，游戏开发的任务就是想办法让静态游戏元素动起来，并提供相应的控制操作。

Android 平台对于游戏开发提供了很好的支持，View、SurfaceView 和 GLSurfaceView 三种视图分别用于开发 2D 游戏和 3D 游戏。三者的继承关系如图 7.1 所示。

```
java.lang.Object
  └android.view.View
      └android.view.SurfaceView
          └android.opengl.GLSurfaceView
```

图 7.1　游戏开发三种视图继承关系

View 类是 Android 平台控件的父类，在游戏开发中游戏界面可以直接继承 View，重写相应方法。SurfaceView 是对 View 的扩展，更适合开发 2D 游戏，但与 View 的工作原理略有不同。GLSurfaceView 主要用于 3D 游戏开发，支持 GPU 加速，涉及 OpenGL 的相关知识。本章主要介绍 View 和 SurfaceView 两种视图的使用。

7.1 游戏开发基础

7.1.1 开发前的思考

在游戏开发前首先要确认的就是游戏脚本、游戏类型，如射击类、竞速类、益智类或是角色控制等。根据游戏的类型，确认是否需要一个主循环。如果游戏不依赖于时间或者仅对用户所做的操作加以反馈，不做任何视觉上的改变，只是等着用户的输入，那么这类游戏不需要主循环。如果是动作类游戏或者带有动画、定时器或任何自动操作的游戏，应该使用主循环。

游戏的主循环需要在自己的线程中运行，避免阻塞主 UI 线程。主循环执行的顺序通常如下：更新状态、接受输入、游戏逻辑、物理响应、播放动画、播放声音、绘制界面等过程。

更新状态就是管理状态的转换。例如，游戏的结束、人物的选择或下一个级别。很多时候需要在某个状态上等待几秒钟，而状态管理应该处理这种延迟，并且在时间过了之后设置成下一个状态。接受输入是指用户的输入，对游戏中相应元素的控制。游戏逻辑是对用户操作的判断。物理响应包括设备反馈、数据持久化等操作。播放动画与声音是指展现游戏界面更迭，声音可以选择使用 SoundPool（播放音效）或者 MediaPlayer（播放背景音乐）。重绘界面以实现游戏的动态。

除了上述内容之外，在开发游戏之前，还要确定是做 3D 还是 2D。2D 游戏有一个较低的学习曲线，适合入门学习，3D 游戏需要更深入地学习数学和物理知识，还需要会使用 3D Studio 和 Maya 那样的建模工具。

7.1.2 关于刷屏

游戏对于用户是动态的，游戏对于开发者是静态的。如何使得静态的游戏元素看起来在不停地运动变化呢？学习过计算机动画或视频制作的读者应该知道视觉暂留效应，视觉暂留时间约为 0.05～0.2s。此处刷屏的概念是指，在上述时间内，将游戏画面重新绘制，新界面会覆盖原来的界面。

具体实现刷屏的方法是绘制一张与游戏可视区一致的背景图片，如果没有背景也可以绘制相应的颜色覆盖原来的界面。

7.1.3 屏幕坐标系

在 Android 系统中，屏幕的左上角是坐标系统的原点(0,0)坐标。水平向右延伸是 X 轴正方向，竖直向下延伸是 Y 轴正方向，如图 7.2 所示。游戏元素的位置都由坐标来确定，以(0,0)坐标为左上角顶点，以(screenWidth，screenHeight)为右下顶点形成的矩形区域是可视区，该区域的元素可以呈现给用户，其他区域的图片不可见。

为了在屏幕中的合适位置绘制图形，需要使用屏幕的宽和高作为参考，来确定绘制图形的位置。要获得屏幕的宽和高，首先从 Activity 对象中获得 WindowManager 对象，然后从 WindowManager 对象中获得 Display 对象，再从 Display 对象中获得屏幕的宽和高，如

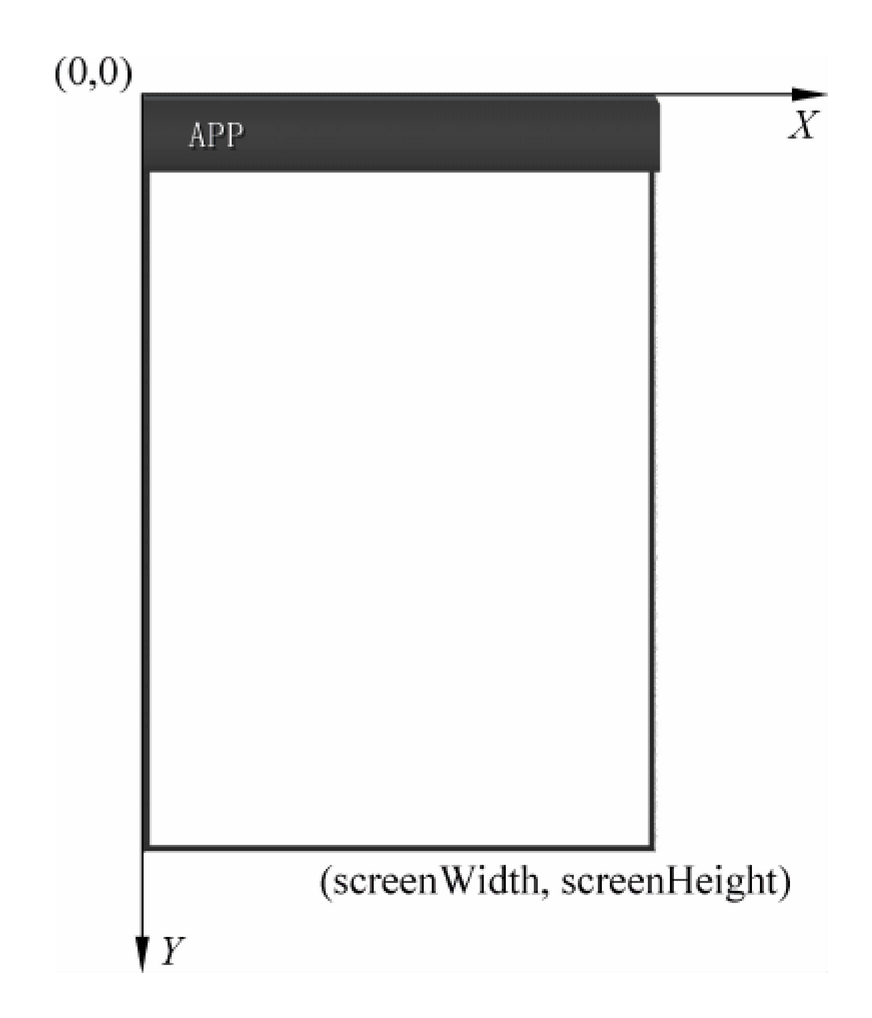


图 7.2　手机屏幕坐标系

代码 07-1 所示。

代码 07-1

```
WindowManager wm = getWindowManager();
Display dis = wm.getDefaultDisplay();
//Android 3.2 之前这样做
Log.i("Tag", dis.getWidth() + " " + dis.getHeight());
/* Android 3.2 之后,提倡下面的方法
Point p = new Point();
dis.getSize(p);
Log.i("Tag", p.toString());
 */
```

在很多游戏中都需要对绘制在屏幕中的元素进行边界的确定。例如,在射击类游戏中需要判断玩家、敌人、子弹等元素的边界位置。边界的判断是对上、下、左、右屏幕边界的判断。如果当前元素的 X 坐标小于 0,则当前视图左越界。如果当前元素的 X 坐标加上宽度大于屏幕的宽,则右越界。如果当前元素的 Y 坐标小于 0,则当前元素上越界。如果当前元素的 Y 坐标加上高度大于屏幕的高,则下越界。

游戏的实现过程其实比较简单,就是不断改变元素的位置坐标,然后重新将它们绘制在屏幕上。这种坐标的位置改变和绘制过程是通过一定逻辑来控制实现的。元素的移动就是通过改变视图坐标位置来实现的。改变了元素的坐标,刷屏,再重新绘制,在视觉上就会感觉到元素在移动。如果元素水平向左移动,则 X 坐标减小;如果元素水平向右移动,则 X 坐标增大。如果元素垂直向上移动,则 Y 坐标减小;如果元素垂直向下移动,则 Y 坐标增大。

7.1.4　横屏和竖屏

手机应用程序默认都支持横竖屏切换,但如果不做处理会出现程序运行异常,所以在游戏开发过程中多数都是锁定屏幕的方向,主要有以下两种方式来实现。

(1) 通过配置文件制定屏幕方向,如代码 07-2 所示。

(2) 通过代码设置屏幕方向,如代码 07-3 所示。

代码 07-2

```
<activity
    android:name = "com.freshen.code.MainActivity"
    android:screenOrientation = "landscape"①
    android:label = "@string/app_name" />
<activity
    android:name = "com.freshen.code.LitActivity"
    android:screenOrientation = "portrait"②
    android:label = "@string/app_name" />
```

MainActivity 采用横屏设置,由属性①设定;LitActivity 采用竖屏设置,由属性②设定。

代码 07-3

```
//设置屏幕为横屏
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
//设置屏幕为竖屏
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

如果应用程序支持横竖屏切换,为了防止切换后重新启动当前 Activity,需要在配置文件中对 Activity 添加 android:configChanges = "keyboardHidden|orientation"属性,并在 Activity 中重写 onConfigurationChanged 方法。

7.1.5 全屏操作

一般应用程序都会带有标题栏和手机状态栏,游戏中为了让界面最大化,可以将标题栏和状态栏隐藏,具体实现可以通过以下两种方式。

(1) 通过配置文件隐藏标题栏和状态栏,如代码 07-4 所示。

(2) 通过代码隐藏标题栏和状态栏,如代码 07-5 所示。

代码 07-4

```
android:theme = "@android:style/Theme.Black.NoTitleBar"①
android:theme = "@android:style/Theme.Black.NoTitleBar.Fullscreen"②
```

隐藏标题栏由①实现,隐藏标题栏、状态栏实现全屏由②实现。

代码 07-5

```
//隐藏标题栏
requestWindowFeature(Window.FEATURE_NO_TITLE);
//隐藏状态栏
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

特别注意使用代码全屏需要将代码放在 setContentView()方法之前执行,否则会抛出异常。

初次进行手机游戏开发,需要注意以下几条建议。

- 游戏中多数元素都需要自己绘制,在设定坐标时一定要对屏幕坐标系有所了解,并谨记后放入的元素有遮盖前面元素的可能。
- 游戏界面发生变化时需要重新绘制,一定要先刷屏,再绘制新元素,否则以前的元素不会自动清空。
- 在游戏元素绘制过程中一定会用到的对象是画布和画笔,需要熟练掌握这两个对象的使用方法。

7.2　绘制游戏元素

绘制游戏元素必须具备三要素:画布、画笔、视图(呈现绘画作品的控件,也就是 View 或 SurfaceView)。当视图创建完成后,需要设置到相应的 Activity 中,最终才能呈现到用户面前。

7.2.1　View 视图

View 类位于 android.view 包中,自定义 View 视图需要继承 View,并提供参数为 Context 类型的构造方法。下面通过项目介绍 View 视图的工作过程。新建项目 part07_2,具体参考随书配套资料。该项目的主 Activity 是 MainActivity.java,不需要布局文件,因为现在的布局视图由继承了 View 的自定义视图实现。核心代码如代码 07-6 所示。

代码 07-6

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //隐藏标题栏
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    //隐藏状态栏
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    //设置该 Activity 使用的视图
    setContentView(new GameView(this));①
}
```

①处代码设定 Activity 的视图为 GameView 的对象。GameView 继承了 View,提供参数为 Context 类型的构造方法(Activity 是 Context 的间接子类),并重写 onDraw 方法,如代码 07-7 所示。

代码 07-7

```
public class GameView extends View {
    Context context;
    //自定义 View 控件,必须提供 Context 参数的构造方法
    public GameView(Context context) {
```

```
            super(context);
            this.context = context;
        }
        //绘制游戏元素需要靠 onDraw 方法
        @Override
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
            Paint paint  = new Paint();
            paint.setColor(Color.RED);
            canvas.drawCircle(50, 100, 20, paint);
        }
    }
```

onDraw 方法是 View 类提供的，用于绘制自定义的图形和图片。参数 canvas 就是一个画布对象，可以直接使用。三要素中画布是直接提供的，视图是自定义的，现在就缺画笔 Paint。Paint paint ＝new Paint()可以直接创建画笔对象。

有过 J2SE 编程经验的读者对于在 Java 中重绘组件应该不陌生，但是在 Android 中的绘制与 Java 中有很大的不同，即便如此，有些思想仍然是可以借鉴的。Android 中绘制图形的方法都是由画布 Canvas 提供的，画笔只能决定颜色、笔触等因素。

7.2.2 Canvas 画布

Canvas 类位于 android.graphics 包中。Android 中绘制图形主要使用 Canvas 提供的各种方法，常用的绘图方法见表 7.1。

表 7.1 常用绘图方法

方法及参数	说　明
drawARGB(int a, int r, int g, int b)	根据 argb 绘制颜色，a 是 alpha 表示透明度
drawArc (RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)	绘制弧线
drawBitmap (Bitmap bitmap, Matrix matrix, Paint paint)	使用矩阵绘制位图
drawBitmap(Bitmap bitmap, float left, float top, Paint paint)	指明坐标绘制位图
drawCircle(float cx, float cy, float radius, Paint paint)	指定坐标绘制圆形
drawColor(int color)	绘制指定颜色
drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	根据两点坐标绘制直线
drawLines(float[] pts, Paint paint)	直线的坐标存放在数组中
drawOval(RectF oval, Paint paint)	绘制椭圆，与矩形外切
drawPath(Path path, Paint paint)	绘制路径
drawPoint(float x, float y, Paint paint)	绘制一点
drawRGB(int r, int g, int b)	绘制 rgb 颜色
drawRect(float left, float top, float right, float bottom, Paint paint)	根据坐标绘制矩形

续表

方法及参数	说　　明
drawRect(RectF rect, Paint paint)	绘制指定圆角矩形
drawRect(Rect r, Paint paint)	绘制指定矩形
drawRoundRect(RectF rect, float rx, float ry, Paint paint)	绘制圆角矩形
drawText(String text, float x, float y, Paint paint)	绘制字符串
drawTextOnPath(char[] text, int index, int count, Path path, float hOffset, float vOffset, Paint paint)	沿指定路径绘制字符串

修改项目 part07_2 中 GameView.java 源文件 onDraw 方法中的代码，添加常用绘图方法，如代码 07-8 所示，参数的含义代码中给出了说明，运行效果如图 7.3 所示。

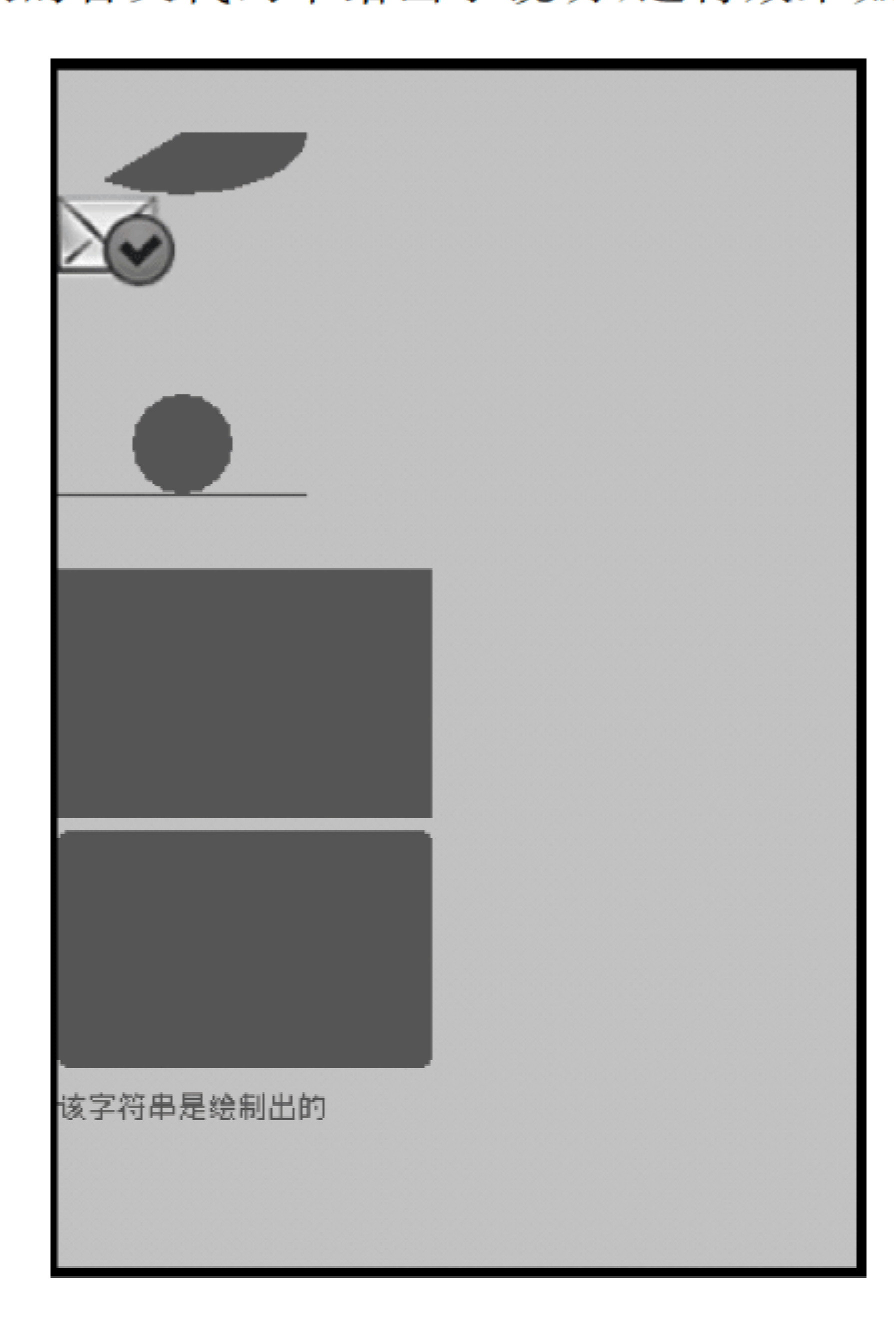


图 7.3　Canvas 绘制基本图形

📖 代码 07-8

```
//绘制游戏元素需要依靠 onDraw 方法
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Paint paint = new Paint();
    paint.setColor(Color.RED);
    //绘制背景色,A 指明透明度
    canvas.drawARGB(100, 150, 150, 150);
    //创建圆角矩形,左上角顶点(0,0),右下角顶点(100,50)
    RectF rf = new RectF(0,0,100,50);
```

```
        //在指定的圆角矩形中绘制内切弧,弧度从 0 度开始,130 度结束,与中心连接
        canvas.drawArc(rf, 0, 130, true, paint);
        //得到 res 包中的位图
        Bitmap bp = BitmapFactory.decodeResource(getResources(), R.drawable.set_02);
        //绘制位图,位图左上角顶点(0,50)
        canvas.drawBitmap(bp, 0, 50, paint);
        //绘制圆形,圆心(50,150),半径 20
        canvas.drawCircle(50, 150, 20, paint);
        //绘制直线,起点坐标(0,170),终点坐标(100,170)
        canvas.drawLine(0, 170, 100,170, paint);
        //绘制矩形,左上角顶点坐标(0,200),右下角顶点(150,300)
        canvas.drawRect(0, 200, 150, 300, paint);
        //创建矩形
        RectF rf2 = new RectF(0,305,150,400);
        //绘制圆角矩形,弧度 5
        canvas.drawRoundRect(rf2, 5, 5, paint);
        //绘制字符串
        canvas.drawText("该字符串是绘制出的", 0, 420, paint);
    }
```

除了表 7.1 所列举的用于绘制图形图像的方法,Canvas 还提供了表 7.2 列举的可以实现状态改变的方法。

表 7.2　改变 Canvas 状态的方法

方　法　名	说　　明
translate (float dx, float dy)	x 轴移动 dx 大小,y 轴移动 dy 大小
rotate(float degrees)	围绕坐标原点,旋转 degrees 度
rotate(float degrees, float px, float py)	围绕 px,py 旋转 degrees 度,常用
scale(float sx, float sy)	以原点为参考,x 轴放缩 sx 倍,y 轴放缩 sy 倍
scale(float sx, float sy, float px, float py)	以 px,py 为参考点,x 轴放缩 sx,y 轴放缩 sy,常用

代码 07-9

```
/* 以下代码测试镜像 */
canvas.save();         //改变画布状态前,需要先保存画布状态        ┐
//画布放缩,参考坐标为 (位图宽度的 2 倍,50),x 方向镜像,y 保持      ├②
canvas.scale(-1, 1, bp.getWidth() * 2, 50);                       │
canvas.drawBitmap(bp, bp.getWidth() * 2, 50, paint);              ┘
canvas.restore(); //恢复画布状态                    ┐
/* 以下代码测试放大 2 倍 */                          │
canvas.save();                                       ├③
canvas.scale(2, 2, bp.getWidth() * 2,50);           │
canvas.drawBitmap(bp, bp.getWidth() * 2, 50, paint);│
canvas.restore();                                    ┘
/* 以下代码测试旋转 */                               ┐
canvas.save();                                       ├④
canvas.rotate(90f, bp.getWidth() * 4.5f, 50f);      │
canvas.drawBitmap(bp, bp.getWidth() * 4, 50, paint);┘
canvas.restore();
```

对照图 7.4 和代码 07-9，分析画布状态改变后，参数的意义，以下分别对②～④处代码进行解析。

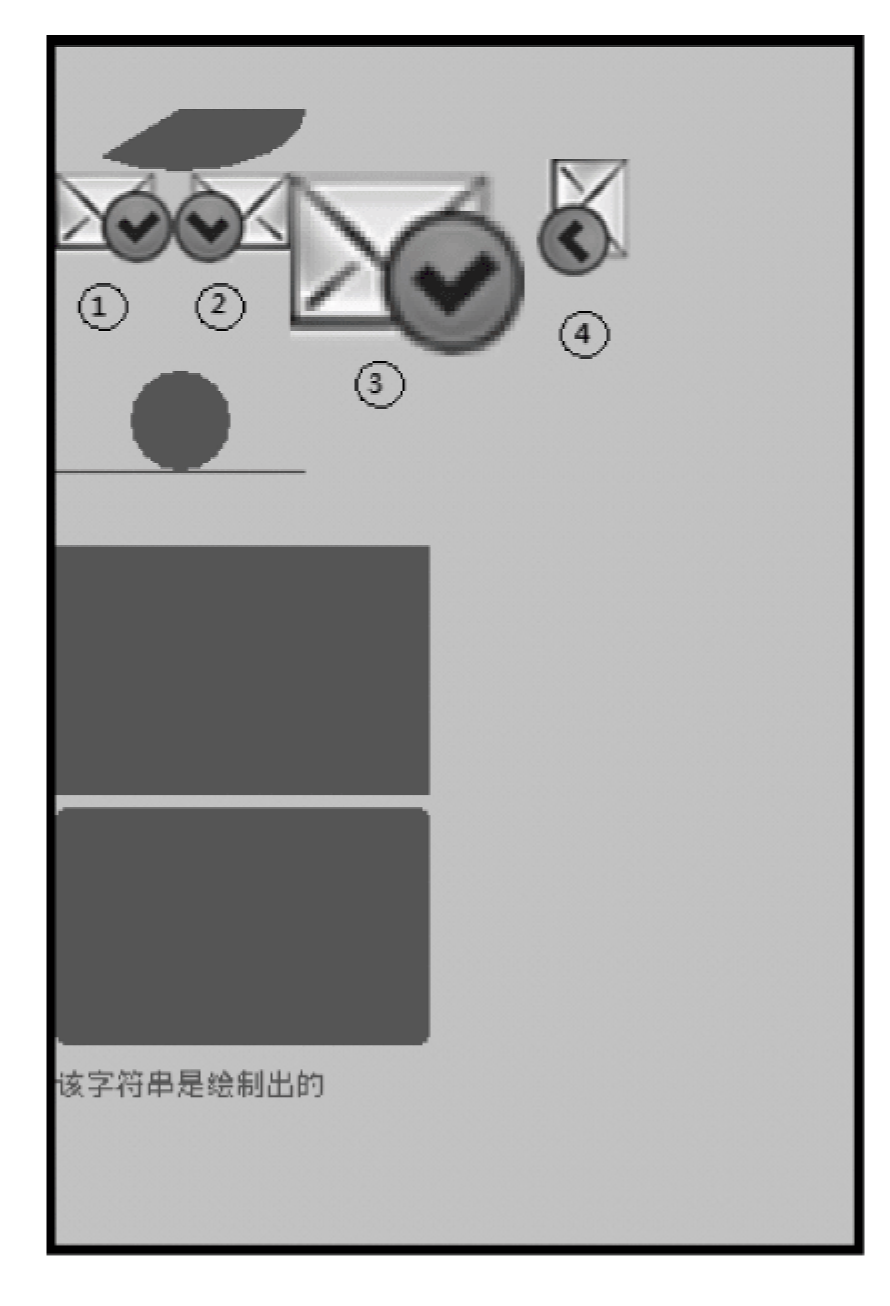


图 7.4 画布状态改变的绘制效果

第②处代码 canvas.scale(－1，1，bp.getWidth()＊2，50)参数的含义如下。

(1) 第一个参数 $sx=-1$，负数表明 x 轴方向会发生镜像，绝对值为 1 说明大小不变。

(2) 第二个参数 $yx=1$，y 轴保持不变。

(3) 第三个参数 $px=$ bp.geWidth＊2，x 轴的参考点为位图 bp 的宽度的 2 倍，这个坐标是相对于画布状态为改变时。

(4) 第四个参数 $py=50$，y 轴参考点为 $50px$，这是相对于画布状态未改变时。

画布状态改变，意味着坐标会发生改变。绘制位图 bp②时的参考点是(bp.getWidth()＊2，50)，x 方向已经是镜像了，所以本该水平向右绘制的位图，现在水平向左绘制，y 方向不变。

第③处代码 canvas.scale(2，2，bp.getWidth()＊2，50)参数的含义如下。

(1) 第一个参数 $sx=2$，x 轴放大为原来 2 倍。

(2) 第二个参数 $yx=1$，y 轴放大为原来 2 倍。

(3) 第三个参数 $px=$ bp.geWidth＊2，x 轴的参考点为位图 bp 的宽度的 2 倍，这个坐标是相对于画布状态为改变时。

(4) 第四个参数 $py=50$，y 轴参考点为 $50px$，这是相对于画布状态未改变时。

第③处绘制位图依然参考(bp.getWidth()＊2，50)，但是现在绘制位图的 x 方向依然是水平向右，但大小会扩大 2 倍；y 方向水平向下，但大小扩大 2 倍。

第④处代码 rotate(90f，bp.getWidth()＊4.5f,50)参数的含义如下。

(1) 第一个参数 $degrees=90$，说明画布顺时针旋转 90 度。

（2）第二个参数 px= bp.getWidth() * 4.5f，x 轴的参考点为位图 bp 宽度的 4.5 倍，想与③号图拉开一些距离。

（3）第三个参数 py=50，y 轴坐标为 50。

画布旋转后，坐标系跟着旋转，绘制出来的位图又要以正常坐标系观看，所以位图④是旋转 90 度的效果。

另外值得注意的是，每次改变画布状态时，需要提前使用 canvas.save()保存画布当前状态，改变状态，绘制图形后需要 canvas.restore()恢复画布的状态。这样，只有在两行代码之间的绘图受到影响，否则画布上的所有元素都将受到影响。

画布的上述方法在线程中不断调用，就可以实现动画的效果，在动画的相关章节会有介绍。Canvas 还有一类方法是 clip×××()，用于在画布上设置剪贴区，表 7.3 列举了常用方法及说明。

表 7.3　Canvas 常用剪贴区方法

方　法　名	说　　明
clipPath(Path path, Region.Op op)	剪切路径，根据 op 参数进行运算
clipRect(Rect rect, Region.Op op)	剪切矩形，根据 op 参数进行运算
clipRect(int left, int top, int right, int bottom)	剪切矩形
clipRegion(Region region, Region.Op op)	剪切 Region 区域，根据 op 参数进行运算

下面新建项目 part07_3，演示剪切区、Path、Region 与 Region.OP 运算。剪切区域可以显示，其他区域隐藏，多个剪切区之间可以通过 Region.OP 运算。代码 07-10 在画布上设定了两个矩形剪切区，通过 Region.Op.UNION 进行运算（即两个区域并集），并绘制了一张位图，只有在剪切区域的部分可以显示，如图 7.5 所示。

代码 07-10

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.save();
    //设置剪切区
    canvas.clipRect(0, 0, 80, 80);
    Rect rect = new Rect(40,40,100,100);
    //设置剪切区，运算规则为 Region.Op.UNION
    canvas.clipRect(rect, Region.Op.UNION);
    //绘制背景
    canvas.drawBitmap(map, 0, 0, paint);
    canvas.restore();
}
```

多个剪切区之间的运算规则详见表 7.4。

表 7.4 Region.OP 操作

剪切区运算方式	说　明
Op.DIFFERENCE	A 区域和 B 区域的差集范围，即 A－B，只有在此范围内的绘制内容才会被显示
Op.REVERSE_DIFFERENCE	B 区域和 A 区域的差集范围，即 B－A，只有在此范围内的绘制内容才会被显示
Op.INTERSECT	A 区域和 B 区域的交集范围，只有在此范围内的绘制内容才会被显示
Op.REPLACE	B 区域将全部进行显示，如果和 A 有交集，则将覆盖 A 的交集范围
Op.UNION	A 区域和 B 区域的并集范围，即两者所包括的范围的绘制内容都会被显示
Op.XOR	A 区域和 B 区域的补集范围，只有在此范围内的绘制内容才会被显示，见图 7.6



图 7.5　剪切区 UNION 运算

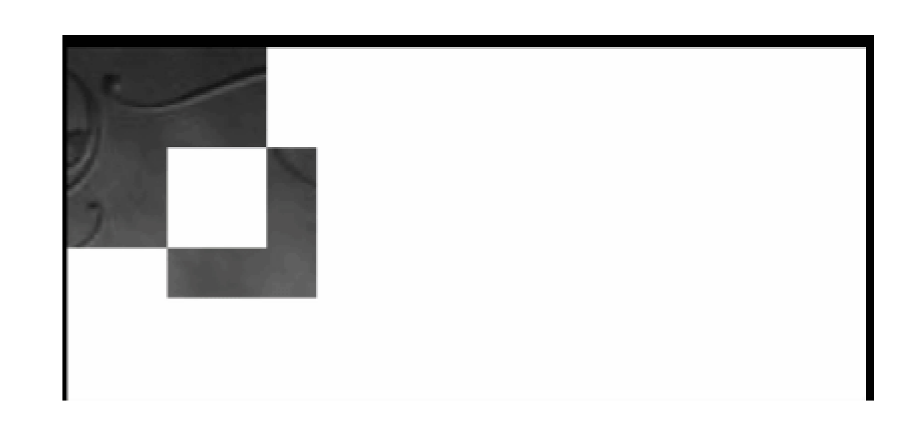


图 7.6　剪切区 XOR 运算

剪切区支持简单图形的剪切，如圆形、矩形，也支持剪切路径 Path 和 Region 区域。剪切 Path，需要先创建 Path 对象。Path 可以作为简单的线条路径，也可以是组合图形的路径。在添加图形时通过 Direction.CW 或 Direction.CCW 分别指明顺时针方向还是逆时针方向。剪切区 Region 相当于多个矩形的组合，代表这些矩形组成的区域，这些矩形通过 Region.OP(见表 7.4)指明运算规则。演示代码见 07-11，运行效果如图 7.7 所示。

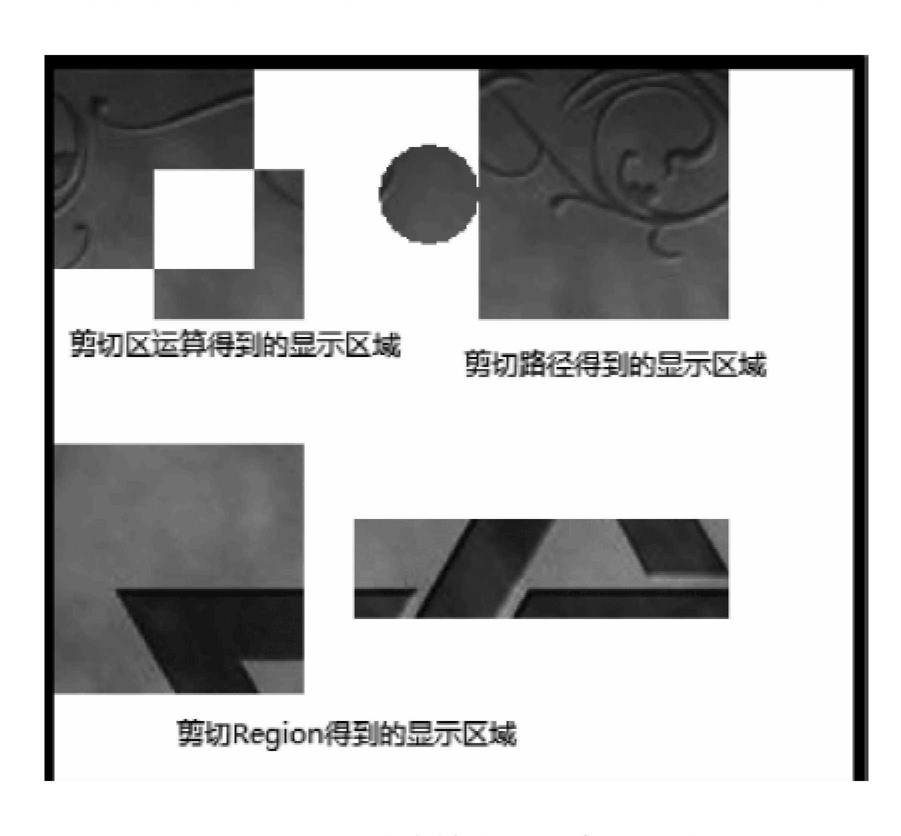


图 7.7　不同剪切区类型对比

📖 **代码 07-11**

```
//创建路径
Path path = new Path();
//在该路径上添加图形
path.addCircle(150, 50, 20, Direction.CW);
path.addRect(170, 0, 270, 100, Direction.CW);
//剪切路径区域,采用并集运算
canvas.clipPath(path,Region.Op.UNION);

//创建 Region 区域
Region region  = new Region(0,150,100,250);
region.op(120, 180, 270, 220, Region.Op.UNION);
//剪切 Region 区域
canvas.clipRegion(region, Region.Op.UNION);
```

7.2.3 Paint 画笔

虽然画布具有很多功能，相对而言，画笔的功能要简单一些。画笔 Paint 类位于 android.graphics 包中，常用的方法见表 7.5。

表 7.5 Paint 类常用方法说明

方　法　名	说　　明
getAlpha()	获取画笔的透明度
getColor()	获取画笔当前颜色
measureText(String text)	测量 text 所占长度
setARGB(int a, int r, int g, int b)	设置透明度及颜色，取值为 0～255
setAlpha(int a)	设置透明度
setAntiAlias(boolean aa)	设置画笔是否具有抗锯齿功能，比较费资源
setColor(int color)	设置颜色
setStrokeWidth(float width)	设置画笔粗细
setStyle(Paint.Style style)	设置画笔样式，默认绘制都是填充，Style. STROKE 可以绘制空心图形
setTextSize(float textSize)	绘制文本时的字体大小

新建项目 part07_4，GameView.java 源代码中 onDraw 方法如代码 07-12 所示，运行效果如图 7.8 所示。

📖 **代码 07-12**

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Paint paint  = new Paint();
    paint.setColor(Color.RED);
    //绘制圆形
    canvas.drawCircle(50, 50, 20, paint);
    canvas.drawText("无抗锯齿", 20, 80, paint);
    //画笔抗锯齿
```

```
        paint.setAntiAlias(true);
        canvas.drawCircle(150, 50, 20, paint);
        canvas.drawText("抗锯齿", 120, 80, paint);
        //只是线条,不填充
        paint.setStyle(Style.STROKE);
        canvas.drawCircle(50, 150, 20, paint);
        //线条粗细
        paint.setStrokeWidth(7);
        canvas.drawCircle(150, 150, 20, paint);
        //绘制文字
        paint.setStrokeWidth(1);
        canvas.drawText("文字宽度" + paint.measureText("文字宽度"),
                50,200, paint);
        //字体大小
        paint.setTextSize(22);
        canvas.drawText("文字宽度" + paint.measureText("文字宽度"),
                150,200, paint);
    }
```

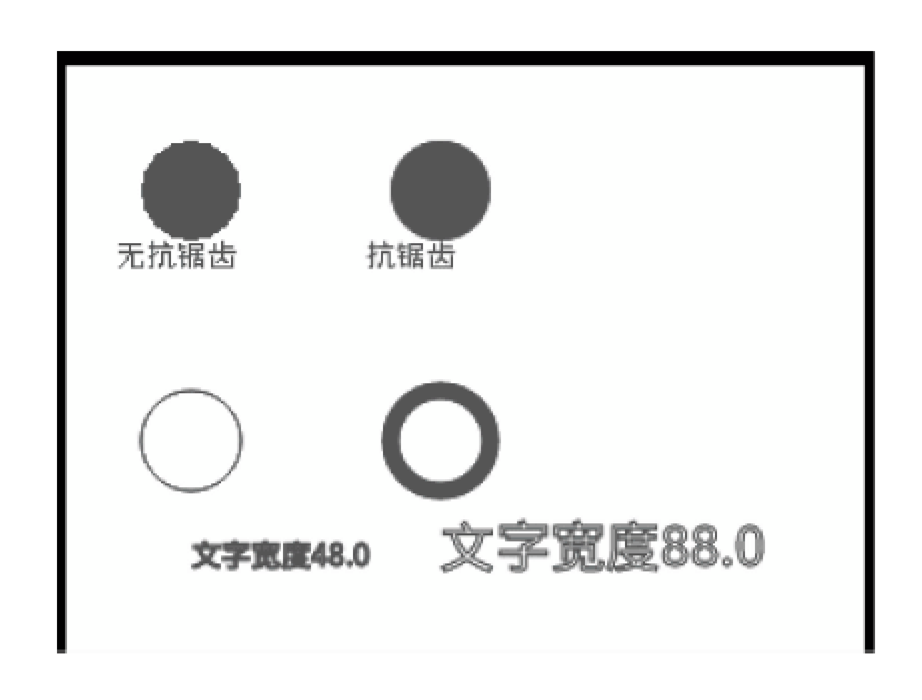


图 7.8　不同笔触的效果

7.2.4　SurfaceView 视图

SurfaceView 继承了 View,但是它的视图绘制机制不同于 View。前面的项目演示了 View 中的方法 onDraw 会被自动调用,写在该方法中的绘图程序得到执行。SurfaceView 虽然继承了该方法,但 SurfaceView 不会自动调用该方法,因此使用 SurfaceView 视图,一般需要自定义绘图方法,然后主动调用。

SurfaceView 采用 SurfaceHolder 对象管理视图的创建、更改和销毁,具体来说,SurfaceHolder 是采用 SurfaceHolder.Callback 中的三个方法来实现 SurfaceView 视图的创建、更改和销毁的。因此,自定义 SurfaceView 视图时,需要继承 SurfaceView,并实现 SurfaceHolder.Callback 接口。

SurfaceHolder.Callback 接口中的三个方法及其作用见表 7.6。

表 7.6　SurfaceHolder.Callback 接口中的方法

方　法　名	说　　明
surfaceCreated	视图创建时调用,用于调用绘图方法、启动线程等操作
surfaceChanged	视图发送改变时调用
surfaceDestroyed	视图销毁时调用,用于停止线程等操作

新建项目 part07_5,演示 SurfaceView 视图的应用流程,完整代码请参考随书配套资料。对于采用 SurfaceView 视图的游戏开发三要素,画笔是直接创建的,视图是继承 SurfaceView 自定义的,画布如何得到呢？在采用 View 视图时,onDraw 方法的参数便是画布的对象,可以直接使用,而在 SurfaceView 中绘图方法需要自定义,此时的画布可以通过 SurfaceHolder 对象获取。该项目 GameView.java 源代码如代码 07-13 所示。

代码 07-13

```
public class GameView extends SurfaceView implements Callback{
    Context context;
    SurfaceHolder holder;          //SurfaceView 的管理器
    Canvas canvas;                 //声明画布和画笔
    Paint paint;
    //自定义 View 控件,必须提供 Context 参数的构造方法
    public GameView(Context context) {
        super(context);
        this.context = context;
        holder = getHolder();          //初始化 holder
        holder.addCallback(this);  //给 holder 添加回调接口
        paint = new Paint();           //创建画笔对象
    }
    //SurfaceView 发送变化时调用
    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
    }
    //SurfaceView 创建时调用
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        //调用自定义绘图方法
        myDraw();
    }
    //SurfaceView 销毁时调用
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
    }
    //自定义绘图方法
    public void myDraw(){
        //获取画布对象
        canvas = holder.lockCanvas();
        if(canvas!= null){
            //游戏元素的绘制
```

```
            canvas.drawCircle(50, 50, 20, paint);
            //所有绘制程序都执行完成后,需要释放canvas
            holder.unlockCanvasAndPost(canvas);
        }
    }
}
```

在自定义绘图方法中,获取画布通过 holder. lockCanvas()。该方法在 SurfaceView 创建后会得到画布,所以该方法需要在 surfaceCreated 方法中调用。如果返回 null,表明当前视图没有创建成功或暂时不可用。画布使用完后通过 holder. unlockCanvasAndPost (canvas)方法释放,此时画布的修改将呈现到 SurfaceView 中。

在游戏开发前,对于选择 View 还是 SurfaceView 视图的建议如下。

- View 视图会自动调用 onDraw 方法,界面需要改变时通过 invalidate 方法或 postInvalidate 方法(在子线程中可以用)通知 View 重新调用 onDraw 方法,即可达到重新绘制界面的目的。
- SurfaceView 虽然通过继承得到了 onDraw 方法,但不会自动调用,所以 SurfaceView 通常会自定义绘图方法,然后通过线程或在需要改变时,直接调用该绘图方法。
- View 多用于棋牌类游戏开发,此类游戏的界面只有在用户触发某个游戏元素时才会发生改变,绘图方法不需要频繁调用。
- SurfaceView 多用于射击竞速类游戏开发,此类游戏即使用户不操作,界面也需要不停地重新绘制,一般可以由线程来调用自定义的绘图方法。

7.3 游戏元素的控制

在手机、平板电脑等设备中对游戏元素的控制只能借助按键和触屏,View 类中已经封装了这两类事件监听的方法,只要重写相应的监听方法就可以实现对游戏元素的控制。目前,智能手机已经逐渐放弃了物理按键的使用,触屏事件的监听逐渐成为手机交互与控制的主要方式,触屏事件、触屏手势也成为游戏控制的主要方式。

7.3.1 按键监听

所谓按键指的是手机的物理按键。View 类中有按键监听相关的方法是 onKeyDown 和 onKeyUp,分别监听键按下键和抬起。手机上对按键的监听与 J2SE 开发中对 PC 键盘的监听类似,可以通过按键的虚拟代码判断是哪个键被触发。

(1) public boolean onKeyDown (int keyCode, KeyEvent event):按键被按下时会不停地响应,keyCode 是被按下键的虚拟代码,event 是键盘事件。

(2) public boolean onKeyUp (int keyCode, KeyEvent event):按键抬起的瞬间响应,keyCode 是被按下键的虚拟代码,event 是键盘事件。

新建项目 part07_6,使用方向键控制 View 视图中的小球移动。MainActivity. java 是主 Activity,GameView. java(代码 07-14)继承 View 视图,MyCircle. java(代码 07-15)是自

定义的图形元素，可以看作是一个游戏元素。

代码 07-14

```
//绘制游戏元素需要依靠 onDraw 方法
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    //每次重绘时，都会先刷颜色，这是刷屏的一种方式
    canvas.drawColor(Color.WHITE);
    //绘制游戏元素
    mc.draw(canvas, paint);
}
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    //判断按键
    switch(keyCode){
    case KeyEvent.KEYCODE_DPAD_UP:          //向上按键
        mc.move(0, -5);break;
    case KeyEvent.KEYCODE_DPAD_LEFT:        //向左按键
        mc.move(-5, 0);break;
    case KeyEvent.KEYCODE_DPAD_DOWN:        //向下按键
        mc.move(0, 5);break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:       //向右按键
        mc.move(5, 0);break;
    }
    invalidate();                           //通知 UI 发送改变，onDraw 会自动调用
    return super.onKeyDown(keyCode, event);
}
```

代码 07-14 重写了 onKeyDown 方法，只要键被按下，该方法会不停地被触发，通过判断 keyCode 参数，响应上下左右 4 个方向，执行游戏元素 MyCircle 的 move 方法。使用 invalidate()方法通知系统 UI 发生改变，调用 onDraw 重写绘制，如果是在子线程中通知 UI 发生改变，需要调用 postInvalidate()方法。

需要注意的是，如果需要程序启动后就能响应按键监听，需要在构造方法中设置 setFocusable(true)，即该游戏视图会获得 Activity 上的焦点，只有获得了焦点的控件才能应用按键事件，有键盘操作经验的读者应该能理解这是必需的，如果是触屏监听，该方法不是必需的，因为触屏时，响应控件自然会得到焦点。

代码 07-15

```
public class MyCircle {
    int px,py,pr;
    int color;
    public MyCircle(int px, int py, int pr, int color) {
        super();
        this.px = px;
        this.py = py;
        this.pr = pr;
```

```
            this.color = color;
        }
        //游戏元素把自己绘制在画布上
        public void draw(Canvas canvas,Paint paint){
            //保留画笔的颜色
            int c = paint.getColor();
            //画笔重新上色
            paint.setColor(color);
            //绘制圆形
            canvas.drawCircle(px, py, pr, paint);
            //画笔恢复颜色,这样做的目的在于不影响画笔绘制其他元素时颜色被改变
            paint.setColor(c);
        }
        //游戏元素的移动方法
        public void move(int x,int y){
            this.px + = x;
            this.py + = y;
        }
    }
```

游戏中每个元素都应对应一个类。本项目中的游戏元素比较简单,只是一个圆形,所具备的属性和方法相对简单,如代码 07-15 所示。这些游戏元素一般都会拥有的方法是 draw(自己绘制自己)和 move(移动,除非该游戏元素不需要移动)。

手机在按键时,调用了该元素的 move 方法,改变了该元素的坐标,然后通过 invalidate,通知 UI 调用 onDraw 重新绘制,在 onDraw 方法中,首先执行 canvas.drawColor(Color.WHITE),将画布背景涂成白色,然后调用游戏元素的绘制方法,重新绘制坐标改变后的图形,在用户眼中会感觉游戏元素在移动。另外,在绘制游戏元素时对画笔颜色的处理是为了防止在此处改变颜色,会影响其他元素的绘制,所以在改变画笔颜色时一般都会先保存画笔的状态,使用完后,再恢复画笔状态,这一点比较重要。

7.3.2 触屏监听

触屏操作虽然有很多事件,如单击、抬起、长按、滑动等,但触屏监听方法就只有一个,即 onTouchEvent,通过参数 MotionEvent 来区分不同的触屏操作,如 ACTION_DOWN、ACTION_MOVE、ACTION_UP 等。

public boolean onTouchEvent (MotionEvent event),触屏时响应,event 封装了触屏的不同事件,以及相应的数据,如触屏的坐标等。

在 GameView.java 代码中重写 onTouchEvent 方法,如代码 07-16 所示,实现对触屏的监听。获取触屏坐标,设置给 mc,运行后小球随单击位置发生改变。

代码 07-16

```
//触屏方法
@Override
public boolean onTouchEvent(MotionEvent event) {
    Log.i("Tag", "触屏事件代码: " + event.getAction());
```

```
        //触屏后抬起
        if(event.getAction() == MotionEvent.ACTION_UP){
            //获取触屏点坐标
            int x = (int) event.getX();
            int y = (int) event.getY();
            mc.moveTo(x, y);
        }
        invalidate();
        return true;
    }
```

7.3.3 线程

像射击类、RPG 类游戏，界面的更新并不是在某些事件的触发下进行的，如动态的花草、流水、云雾、地图的滑动等，在这些情景下需要使用子线程，不停地调用绘图方法，不断刷新游戏界面，使用户感觉游戏界面动感十足。

新建项目 part07_7，MainActivity.java 是主 Activity 源文件，GameSurfaceView.java 是采用 SurfaceView 的游戏视图（参考代码 07-17，其他源文件查看随书配套资料），MyCircle.java 是游戏元素类。

代码 07-17

```
public class GameSurfaceView extends SurfaceView implements Callback,Runnable{
    Context context;
    SurfaceHolder holder;              //SurfaceView 的管理器
    Canvas canvas;                     //声明画布和画笔
    Paint paint;
    boolean flag = true;               //控制线程的标志位
    //游戏元素
    MyCircle mc;
    //自定义 View 控件,必须提供 Context 参数的构造方法
    public GameSurfaceView(Context context) {
        super(context);
        this.context = context;
        holder = getHolder();          //初始化 holder
        holder.addCallback(this);      //给 holder 添加回调接口
        paint = new Paint();           //创建画笔对象
        //创建游戏元素
        mc = new MyCircle(50,50,20,Color.GREEN,this);
    }
    //SurfaceView 发送变化时调用
    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
    }
    //SurfaceView 创建时调用
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        flag = true;
```

```
        //启动线程
        new Thread(this).start();
    }
    //SurfaceView 销毁时调用
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        flag = false; //SurfaceView 视图销毁时,子线程停止
    }
    //自定义绘图方法
    public void myDraw(){
        //获取画布对象
        canvas = holder.lockCanvas();
        if(canvas!= null){
            //刷屏
            canvas.drawColor(Color.WHITE);
            //游戏元素的绘制
            mc.draw(canvas, paint);
            //所有绘制程序都执行完毕后,需要释放 canvas
            holder.unlockCanvasAndPost(canvas);
        }
    }
    //线程
    @Override
    public void run() {
        //标志位为 true,线程一直运行
        while(flag){
            long st = System.currentTimeMillis();
            //游戏逻辑,以及绘图代码
            myDraw();
            mc.move();
            long et = System.currentTimeMillis();
            //实现定时刷新游戏界面,间隔 80ms
            if(et - st < 80){
                try {
                    Thread.sleep(80 - (et - st));
                } catch (InterruptedException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
    //触屏监听
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if(event.getAction() == MotionEvent.ACTION_UP){
            mc.moveTo(event.getX(),event.getY());
        }
        return true;
    }
}
```

代码 07-17 中黑体字部分是子线程的启动、执行、停止三种操作分别对应或执行的方法。上述代码可以看作是对线程使用的一个模板，只需要在 run 方法中修改逻辑即可。操作线程一定要控制好启动、停止的时机。在 surfaceCreated 方法中创建并启动线程，可以保证无论用户是按 Back 键退出，还是按 Home 键退出，再次开启时，线程一定会启动。在 surfaceDestroyed 方法中停止线程是必需的。线程的启动、停止是否在合理的方法、时间执行，可以通过 Back 键和 Home 键两种方式退出，再启动 APP 检验是否有异常发生来确认。

7.4 位图的使用

前面程序中在使用图片时大多是以 Drawable 资源的方式引用，这种引用是通过 R. drawable. ×××或@drawable/×××引用放入 res/drawable－××××资源文件夹中的图片。这些图片都是项目中的图片资源，如果要引用应用程序之外的图片资源(SD 卡上图片、图库中图片等)就需要使用位图。Android 系统支持的图片格式有 PNG、JPG 和 BMP。PNG 格式的图片可以实现透明图层，在游戏开发中使用较多，JPG 格式的图片可以用作背景。除此之外，Android SDK 中提供了 9patch 工具，用于编辑 PNG 格式的图片，使其支持放缩，编辑后的后缀名为. 9. png。

7.4.1 创建位图

位图类是 Bitmap，位于 android. graphics 包中，创建位图对象可以通过 Bitmap 的静态方法，见表 7.7，也可以通过 BitmapFactory，见表 7.8。

表 7.7 Bitmap 部分静态方法

返回值	方法声明	说明
static Bitmap	createBitmap(Bitmap source, int x, int y, int width, int height)	以 source 为目标，从起点坐标(x,y)开始，复制宽高为 width、height 的位图
static Bitmap	createScaledBitmap (Bitmap src, int dstWidth, int dstHeight, boolean filter)	以 src 为目标，放缩成宽高为 dstWidth、dstHeight 的位图
static Bitmap	Bitmap createBitmap(int width, int height, Bitmap. Config config)	创建宽高为 width、height 的位图

表 7.8 BitmapFactory 部分静态方法

返回值	方法声明	说明
static Bitmap	decodeByteArray (byte[] data, int offset, int length)	将 data 数组中从 offset 下标开始，length 字节长度的内容解析成位图
static Bitmap	decodeFile (String pathName)	将 pathName 对应的文件解析成位图
static Bitmap	decodeResource (Resources res, int id)	将资源文件 drawable 中对应的 R. drawable. id 解析成位图
static Bitmap	decodeStream (InputStream is)	从输入流 is 中解析位图

位图资源不断创建会不停地消耗手机资源，为此 Bitmap 提供了以下两个方法用于控制位图的释放。

（1）final boolean isRecycled ()，返回位图是否被回收。

（2）void recycle ()，回收位图。

7.4.2　位图的操作

位图的操作主要有位图的移动、放缩、旋转等，可以借助 Canvas 类中状态改变的方法实现，前面章节已有介绍；还可以使用 Matrix 类实现。新建项目 part07_8，GameSurfaceView 的绘图方法如代码 07-18 所示。

代码 07-18

```
//自定义绘图方法
public void myDraw(){
    //获取画布对象
    canvas = holder.lockCanvas();
    if(canvas!= null){
        //刷屏
        canvas.drawBitmap(map, 0, 0, paint);
        //创建矩阵
        Matrix mx = new Matrix();
        //矩阵放缩比例,以及参考点坐标
        mx.postScale(0.5f, 0.5f,getWidth()/2,getHeight()/2);
        //以矩阵为范本,绘制位图
        canvas.drawBitmap(map, mx, paint);
        //恢复矩阵
        mx.reset();
        //重设矩阵放缩,并旋转 90 度
        mx.postScale(0.25f, 0.25f,getWidth()/2,getHeight()/2);
        mx.postRotate(90, getWidth()/2,getHeight()/2);
        canvas.drawBitmap(map, mx, paint);
        //所有绘制程序都执行完毕后,需要释放 canvas
        holder.unlockCanvasAndPost(canvas);
    }
}
```

坐标矩阵 Matrix 是一个 3×3 的矩阵，可以完成位图的移动、放缩、旋转、透视等操作，具体方法可以查看 Matrix 的 API，Matrix 类位于 android.graphics 包中。项目运行效果如图 7.9 所示。

7.4.3　9patch 编辑器

9patch 编辑器可以处理 png 格式的图片，生成.9.png 格式的图片。这是 Android 系统所支持的一种特殊的图片格式，用它可以实现部分拉伸。

9patch 编辑器位于安装路径下，如 D:\android-sdk\tools\draw9patch.bat。双击打开该编辑器（注意杀毒软件和防火墙，应选择允许），如图 7.10 所示。选择 File→Open 9 path



图 7.9　Matrix 矩阵变形后的位图

命令导入一张 png 图片。打开编辑界面，如图 7.11 所示。9patch 编辑器比较简单，左边视图是编辑区，右边视图是预览区域。所谓编辑就是在原图的上边缘和左边缘绘制黑线，交叉区域就是将来的放缩区域。

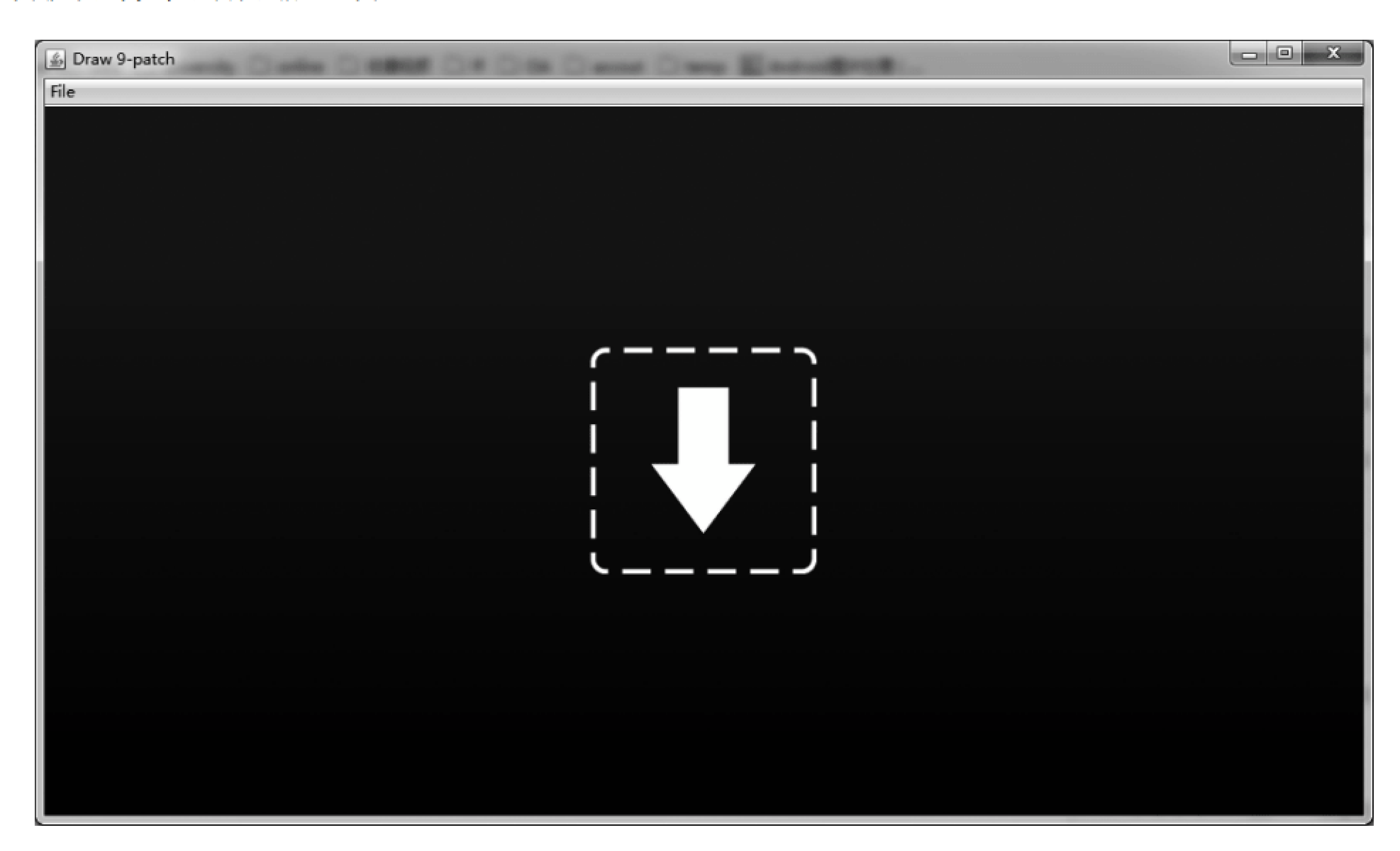


图 7.10　9patch 打开界面

在 part07_8 项目中添加 myDraw9patch 方法，绘制一般 png 位图与 9patch 位图。相关代码参考代码 07-19。绘制 9patch 位图需要用到 NinePatch 类，该类位于 android.graphics 包中。绘制时，使用 NinePatch 的 draw 方法。运行效果如图 7.12 所示。

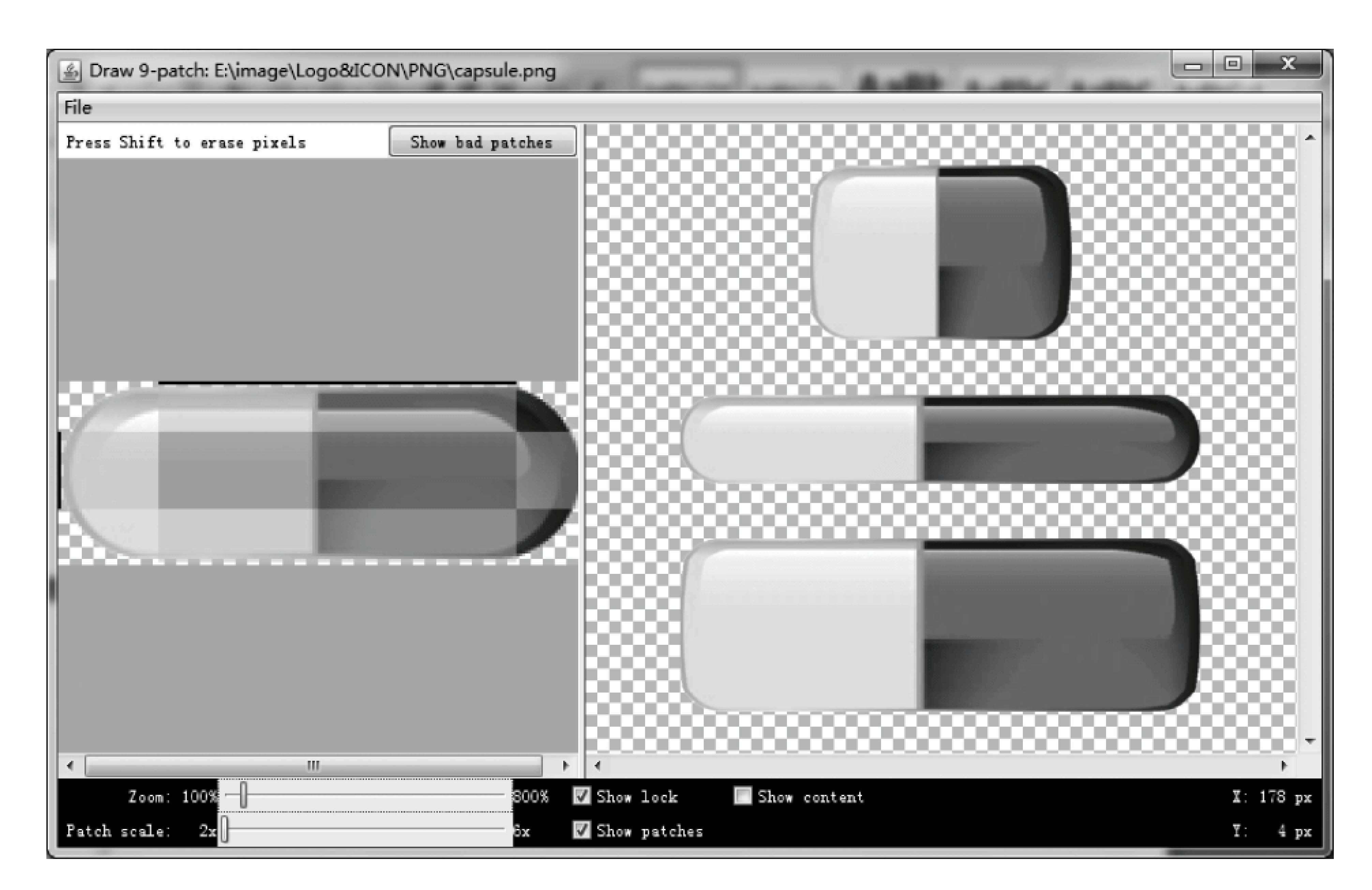


图 7.11 9patch 编辑界面

代码 07-19

```
    //声明属性,对比 png 与 9.png 的放缩效果
    Bitmap png,png9;
    NinePatch np;
 …
        png = BitmapFactory.decodeResource(getResources(), R.drawable.bg);
        png9 = BitmapFactory.decodeResource(getResources(), R.drawable.bg9);
        //创建 9patch 对象
        np = new NinePatch(png9, png9.getNinePatchChunk(), null);
 …
//自定义绘图方法
public void myDraw9patch(){
        //获取画布对象
        canvas = holder.lockCanvas();
        if(canvas!= null){
            //刷屏
            canvas.drawColor(Color.WHITE);
            //绘制原比例图
            canvas.drawBitmap(png, 0, 0, paint);
            //创建放缩目标尺寸
            Rect des = new Rect(0,png.getHeight() + 10,
                    png.getWidth() * 2,10 + png.getHeight() * 3);
            canvas.drawBitmap(png, null, des, paint);
            //原始 9pacth 图尺寸
            canvas.drawBitmap(png9, 200, 0, paint);
            //使用 NinePatch 的绘图方法
```

```
                RectF loc = new RectF(0, png9.getHeight() * 3 + 20, png9.getWidth() * 2, png9.
getHeight() * 5 + 20);
                np.draw(canvas, loc);
                //所有绘制程序都执行完毕后,需要释放 canvas
                holder.unlockCanvasAndPost(canvas);
            }
        }
```

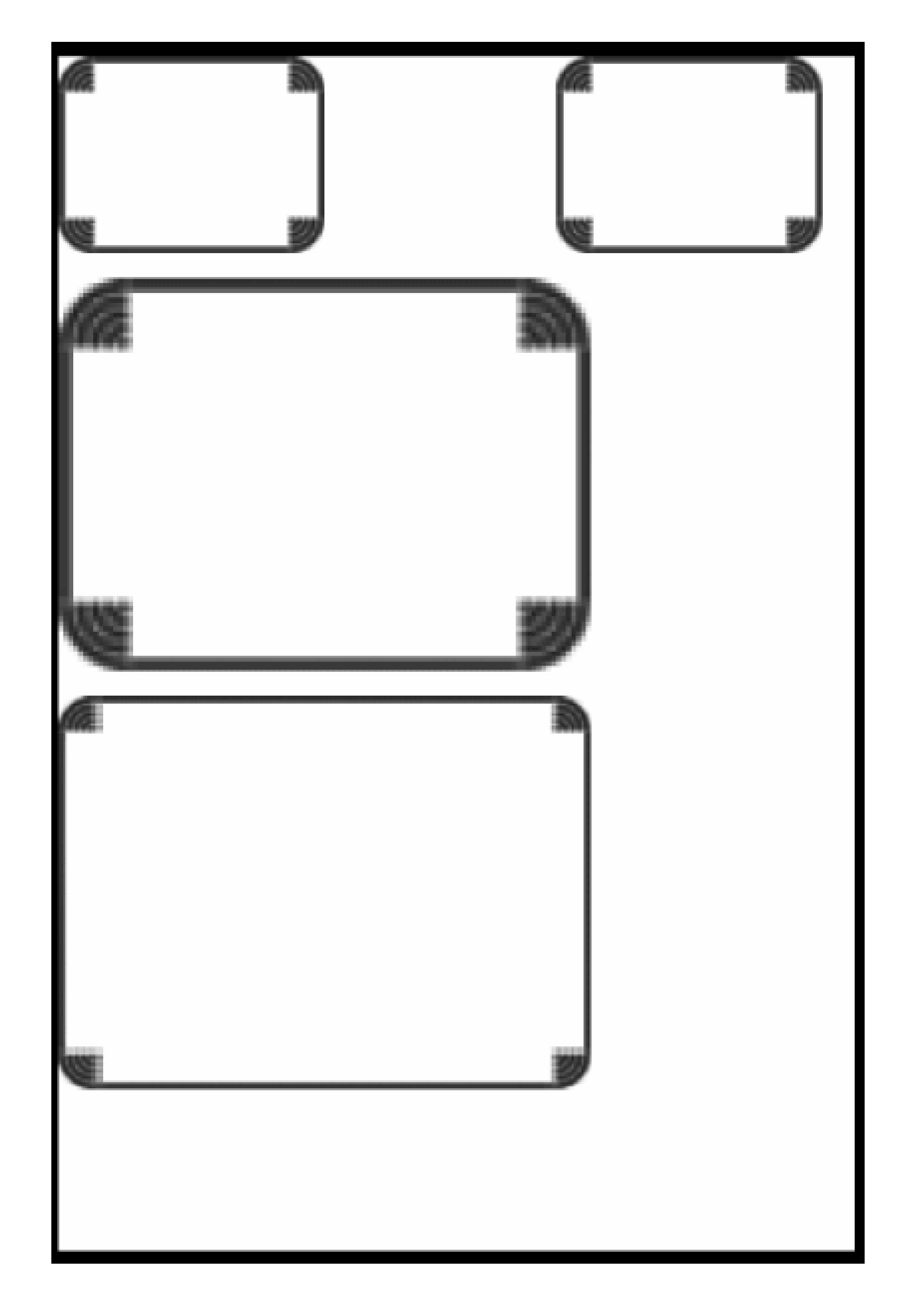


图 7.12 9patch 与 png 对比

7.5 动 画

动画作为游戏中的动态元素是必不可少的。Android 对动画的支持非常强大,可以使用系统提供的动画,也可以使用自定义的动画;可以由代码实现动画,也可以由资源文件定义动画。Android 的动画主要划分为两大类,tweened animation(相当于 Flash 中的补间动画)和 frame-by-frame animation(帧动画)。其中 tweened animation 由 android.view.animation 包中的类实现;frame-by-frame animation 由 AnimationDrawable 类实现,位于 android.graphics.drawable 包中。

7.5.1 tweened animation

Animation 类是 tweened animation 中的一个抽象类,它的实现类有以下几个。

(1) AlphaAnimation,实现透明度动画效果。

(2) RotateAnimation,实现旋转动画效果。

(3) ScaleAnimation,实现放缩动画效果。

(4) TranslateAnimation,实现移动动画效果。

AlphaAnimation 的常用构造方法是 AlphaAnimation (float fromAlpha, float toAlpha),指明开始时的透明度和结束时的透明度,然后通过 setDuration (long durationMillis)(该方法是 Animation 类的)方法设置动画持续时间。0 代表透明,1 代表不透明。

RotateAnimation 旋转动画相对复杂一点,它需要考虑旋转时的参考点问题。RotateAnimation 的构造方法解释如下。

(1) RotateAnimation(float fromDegrees, float toDegrees),默认以 View 左上角顶点为旋转点,x 轴以 fromDegrees 为起始点度数,在规定时间内,x 轴顺时针转动到 toDegrees。如 fromDegrees=0,toDegrees=90;为左上角顶点为旋转点。0 度为起始点,90 度为终点。进行旋转,旋转了 90 度。

(2) RotateAnimation (float fromDegrees, float toDegrees, float pivotX, float pivotY),以(pivotX,pivotY)为旋转点,按照上述规则旋转。

(3) RotateAnimation (float fromDegrees, float toDegrees, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue),其中参数 pivotXType 和 pivotYType 为坐标参考类型,可以取值 Animation. ABSOLUTE(绝对参考)、Animation. RELATIVE_TO_SELF(相对于自己)、Animation. RELATIVE_TO_PARENT(相对于父容器),相对参考时取值 0(代表左边),1(代表右边)。如 RotateAnimation (0, 90, Animation. RELATIVE_TO_SELF, 0. 5f, Animation. RELATIVE_TO_SELF, 0. 5f),即围绕中心点旋转 90 度。

ScaleAnimation 放缩动画与旋转动画的构造方法类似,需要考虑参考点,根据指定的参考坐标,x 轴和 y 轴分别进行放缩。

(1) ScaleAnimation(float fromX, float toX, float fromY, float toY),以原点为参考点,x 轴从 fromX 变化到 toX,y 轴从 fromY 变化到 toY。

(2) ScaleAnimation(float fromX, float toX, float fromY, float toY, float pivotX, float pivotY),以(pivotX,pivotY)为参考点。

(3) ScaleAnimation(float fromX, float toX, float fromY, float toY, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue),参数的含义参考 RotateAnimation 的构造方法。

TranslateAnimation 位移变化动画,在指明开始坐标、结束坐标、持续时间后即可开始动画,其中 Type 类型的说明参考 RotateAnimation 的构造方法。

(1) TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta),x 轴从 fromXDelta 到 toXDelta,y 轴从 fromYDelta 到 toYDelta。

(2) TranslateAnimation (int fromXType, float fromXValue, int toXType, float toXValue, int fromYType, float fromYValue, int toYType, float toYValue),需要指明参考类型。

新建项目 part07_9,GameView 继承 View,使用方向按键触发 4 种动画效果,按键监听方法如代码 07-20 所示,完整代码请查阅随书配套资料。

📖 **代码 07-20**

```
//根据按键决定动画
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
    Log.i("Tag", "keycode = " + keyCode);
    //向上键播放透明度动画
    if(keyCode == KeyEvent.KEYCODE_DPAD_UP){
        this.setBackgroundResource(0);
        aa  = new AlphaAnimation(1, 0);
        aa.setDuration(2000);      //持续时间
        aa.setFillAfter(true);     //动画完成,停留
        startAnimation(aa);
    }else if(keyCode == KeyEvent.KEYCODE_DPAD_LEFT){
        ta = new TranslateAnimation(0, 150, 0, 0);
        ta.setDuration(3000);
        ta.setFillAfter(true);
        startAnimation(ta);
    }else if(keyCode == KeyEvent.KEYCODE_DPAD_DOWN){
        //相对于图片中心
        ra = new RotateAnimation(0, 720, 50 + map.getWidth()/2,50 + map.getHeight()/2);
        ra.setDuration(3000);
        ra.setFillAfter(true);
        startAnimation(ra);
    }else if(keyCode == KeyEvent.KEYCODE_DPAD_RIGHT){
        sa = new ScaleAnimation(1, 2, 1, 2,50 + map.getWidth()/2,50 + map.getHeight()/2);
        sa.setDuration(2000);
        ta.setFillAfter(true);
        startAnimation(sa);
    }
    return super.onKeyUp(keyCode, event);
}
```

使用 Animation 的注意事项如下。

- 动画效果是针对整个 View 视图,而不是 View 视图中的某个元素,谨记这一点。
- Animation 可以直接用于 View 视图,如果用在 SurfaceView 上会发现没有动画效果,原因是 SurfaceView 自带有缓冲(先绘制元素于 Bitmap,然后将 Bitmap 绘制在 Canvas,有的资料称为双缓冲机制)。

Android 系统中的动画还可以通过资源文件生成,按照以下步骤修改项目 part07_9,在布局文件 activity_main.xml 中添加代码 07-21。将上面自定义的 View 视图加入布局文件(黑体字部分),在布局文件中使用自定义的视图,自定义视图类需要实现构造方法 GameView(Context context, AttributeSet attrs)。

📖 **代码 07-21**

```
<com.freshen.code.GameView
    android:id = "@ + id/gameView1"
    android:layout_width = "wrap_content"
```

```
        android:layout_height = "200dp"
        />
    <Button
        android:id = "@ + id/bt_start"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "开始动画"
        android:layout_below = "@id/gameView1"
        />
    <Button
        android:id = "@ + id/bt_end"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "停止动画"
        android:layout_below = "@id/gameView1"
        android:layout_toRightOf = "@id/bt_start"
        />
    <ImageView
        android:id = "@ + id/iv_or"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:src = "@drawable/orage_green"
        android:layout_below = "@id/bt_start"
        />
```

在 res 资源中新建动画资源 myanim.xml,放在 anim 文件夹中,如代码 07-22 所示。在动画资源文件中可以使用 rotate 标签、scale 标签、alpha 标签、translate 标签定义前面讲述的 tween animation 的 4 种动画。另外,set 标签用于定义一组动画。

代码 07-22

```
    <set xmlns:android = "http://schemas.android.com/apk/res/android">
        <rotate
            android:fromDegrees = "0"
            android:toDegrees = "36000"
            android:duration = "5000"
            android:pivotX = "50 % "
            android:pivotY = "50 % "
        />
    </set >
```

MainActivity.java 中的 onCreate 方法如代码 07-23 所示。使用动画工具类 AnimationUtils 可以将动画资源文件解析为动画对象,View 类的子类都会继承到方法 startAnimation()中,用于开始播放动画。项目运行的效果如图 7.13 所示。

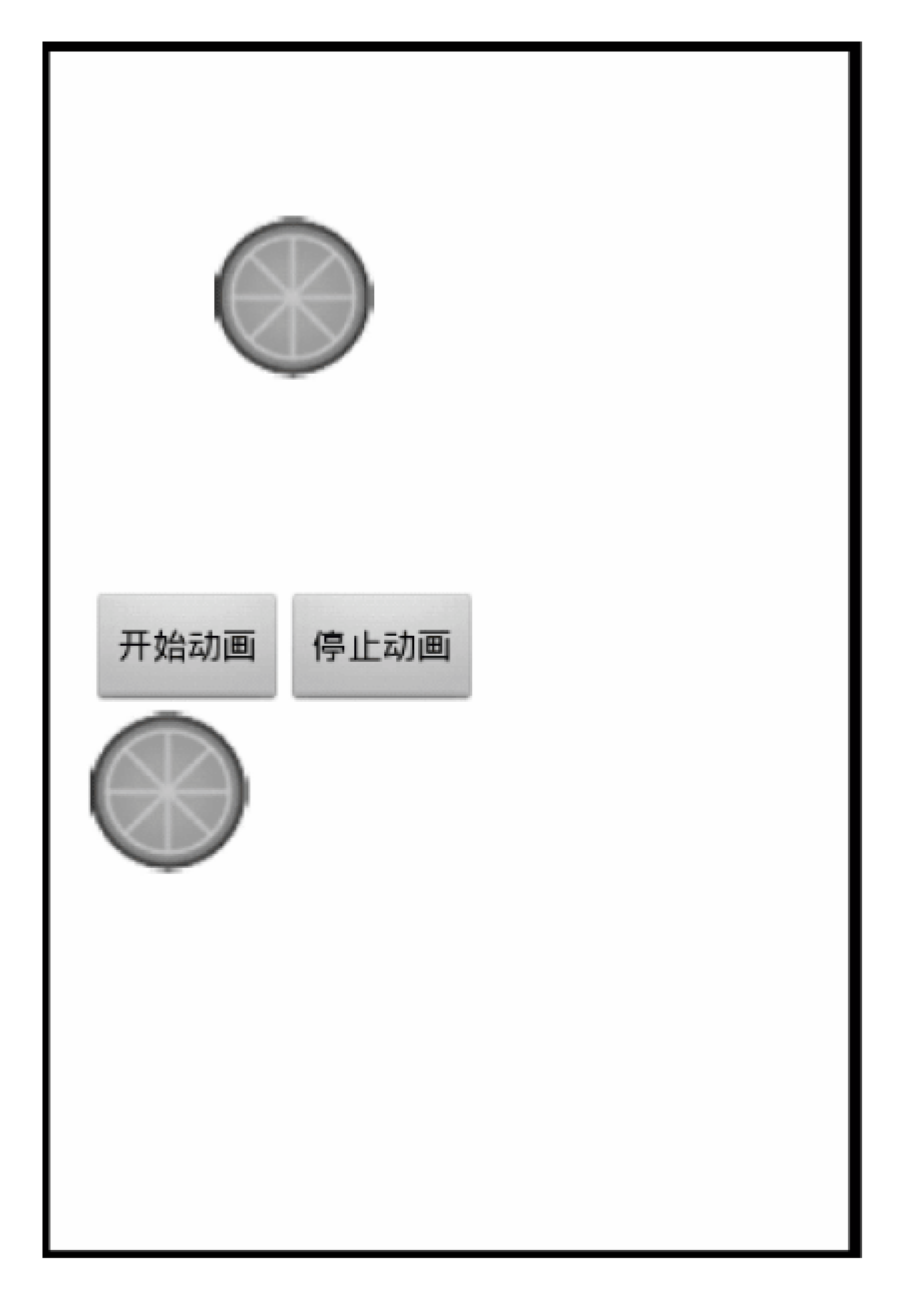


图 7.13　两种动画方式

代码 07-23

```
setContentView(R.layout.activity_main);
b1 = (Button) findViewById(R.id.bt_start);
b2 = (Button) findViewById(R.id.bt_end);
iv = (ImageView) findViewById(R.id.iv_or);
//将资源文件解析为动画
final Animation anim  = AnimationUtils.loadAnimation(this, R.anim.myanim);
anim.setFillAfter(true);
b1.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View arg0) {
        //开始动画
        iv.startAnimation(anim);
    }}
);
b2.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View arg0) {
        iv.clearAnimation();
        anim.cancel();
    }}
);
```

Activity 的上半部分是自定义的 GameView 视图，可以通过方向键触发 4 种动画效果。注意在按向下键时会将焦点移到按钮上，如果想让 GameView 重新得到焦点，只需要对其触屏即可。下半部分是布局文件中的系统控件，通过按钮控制动画的开始和结束。

7.5.2　frame-by-frame animation

帧动画是将多张静态的图片连续播放，利用人眼的视觉暂留效应，给人以动画的感觉。可以通过代码的方式实现，也可以通过资源文件实现。

新建项目 part07_10，采用继承 SurfaceView 视图，线程启动后不断调用 myDraw 方法，每次绘制下一帧，如代码 07-24 所示。该项目采用 8 帧图片，如图 7.14 所示。对于人物行走绘制 8 帧静态图已经比较流畅了，多数手机游戏，有 4 帧图也可以实现。

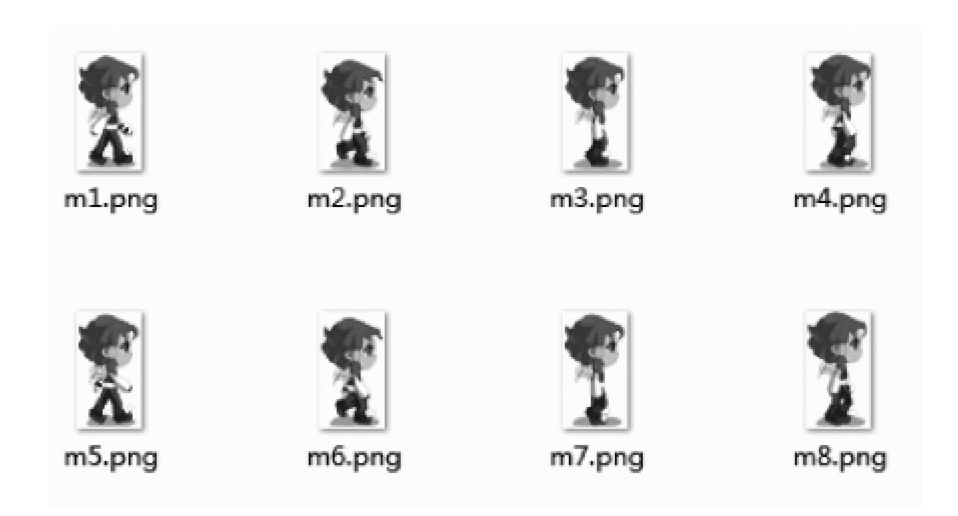


图 7.14　人物行走的 8 帧图

代码 07-24

```
    Bitmap map[] = new Bitmap[8];          //8 帧图片
    int currentFrame = 0;                  //当前播放帧
…
        //初始化 8 帧图片
        for(int i = 0;i<map.length;i++){
            map[i] = BitmapFactory.decodeResource(getResources(), R.drawable.m1 + i);
        }
…
    public void myDraw(){
        //获取画布对象
        canvas = holder.lockCanvas();
        if(canvas!= null){
            canvas.drawColor(Color.WHITE);
            //游戏元素的绘制
            canvas.drawBitmap(map[currentFrame++ % 8], 0, 0, paint);
            //所有绘制程序都执行完毕后,需要释放 canvas
            holder.unlockCanvasAndPost(canvas);
        }
    }
```

从资源文件解析逐帧动画需要先创建资源文件 myanim.xml，位于 res/drawable 文件夹中，如代码 07-25 所示，android:oneshot="false"表示开启循环，如果值为 true 则不循环。使用该资源文件时只需要设定 android:background="@drawable/myanim"属性即可。

代码 07-25

```
<animation - list xmlns:android = "http://schemas.android.com/apk/res/android"
    android:oneshot = "false" >
```

```
        <item android:drawable="@drawable/m1" android:duration="50" />
        <item android:drawable="@drawable/m2" android:duration="50" />
        <item android:drawable="@drawable/m3" android:duration="50" />
        <item android:drawable="@drawable/m4" android:duration="50" />
        <item android:drawable="@drawable/m5" android:duration="50" />
        <item android:drawable="@drawable/m6" android:duration="50" />
        <item android:drawable="@drawable/m7" android:duration="50" />
        <item android:drawable="@drawable/m8" android:duration="50" />
</animation-list>
```

这种由资源文件生成的逐帧动画默认不播放，需要在代码中执行 start，开始播放，停止时执行 stop 即可，如代码 07-26 所示。逐帧动画由类 AnimationDrawable 管理，该类还提供了插入帧的方法 addFrame(Drawable frame, int duration)，可以向其中添加帧。

代码 07-26

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    iv = (ImageView) findViewById(R.id.iv_pl);
    b1 = (Button) findViewById(R.id.bt_go);
    b2 = (Button) findViewById(R.id.bt_stop);
    //获取背景动画
    final AnimationDrawable ad = (AnimationDrawable) iv.getBackground();
    b1.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View v) {
            ad.start();     //开始帧动画
        }
    });
    b2.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            ad.stop();      //结束帧动画
        }
    });
}
```

7.5.3 自定义动画

自定义动画是借助 Canvas 提供的方法，实现某些游戏元素随时间变化的目的。自定义动画时比较关键的内容就是将游戏元素状态数据设定为属性、变量，用户通过相应操作，改变这些属性或变量，重新绘制时根据这些属性和变量值进行，这样就可以完成自定义动画。

新建项目 part07_11，模拟雷电游戏中对飞机的控制，并实现地图的自动滚动，完整代码请参考随书配套资料。项目清单参考表 7.9。

表 7.9 part07_11 源文件清单

代码文件	方法声明	说明
MainActivity.java	onCreate	游戏主 Activity，设定全屏，采用 GameSurfaceView 作为游戏视图
GameSurfaceView.java	GameSurfaceView(Context) surfaceChanged(SurfaceHolder, int, int, int) surfaceCreated(SurfaceHolder) surfaceDestroyed(SurfaceHolder) myDraw() run() onTouchEvent(MotionEvent)	构造方法； SurfaceView 变化监听方法； SurfaceView 创建监听方法，创建游戏元素，启动线程； SurfaceView 销毁监听方法； 自定义绘图方法，绘制地图和飞机； 线程方法，调用地图和飞机的 move 方法； 触屏改变飞机方向
BackMap.java	BackMap(GameSurfaceView) draw(Canvas, Paint) move()	构造方法，需要传入 GameSurfaceView 是为了获取屏幕宽高；自定义绘图方法；地图移动方法
Airplane.java	Airplane(GameSurfaceView) draw(Canvas, Paint) move() makeDir(int, int) getDir() setDir(Direction)	构造方法，同 BackMap； 自定义绘图方法； 移动方法； 确定方向方法，根据传入的坐标确定方向； 获取方向； 直接设定方向

注意，如果要改变飞机的位置需要触屏，当抬起时飞机将不移动；由于地图是一直向左滚动的，整体上会感觉飞机在向右飞行。运行效果如图 7.15 所示。上述项目再添加敌方飞机，实现发射子弹，完成碰撞检测就可以用来模拟雷电了(后面章节会解决游戏背景音乐和音效的问题)。感兴趣的读者可以自己尝试设计开发类似雷电的游戏。



图 7.15 模拟雷电游戏场景

7.5.4 剪切区动画

剪切区动画可以相对于逐帧动画来分析。在使用逐帧动画时，每一帧都对应一张静态图，因此需要准备多张动画所用图片，这样会使得游戏素材增多，安装文件变大。剪切区动画可以实现将人物的分帧动作都放在一张图上，如图 7.16 所示，然后在需要显示区域使用



图 7.16 人物行走位图

剪切区即可。

新建项目 part07_12,演示剪切区动画,该项目框架与 part07_11 基本一致,只是在播放人物行走时使用剪切区。游戏人物源代码文件是 player.java,其中关键部分如代码 07-27 所示。运行该项目,可以通过方向键控制人物的行走,同时播放图 7.16 中的不同人物造型。

代码 07-27

```
…
    /*声明属性*/
    //移动方向
    public enum Direction{DOWN,LEFT,RIGHT,UP}
    //人物位图
    Bitmap lhch;
    //人物坐标,行走分速度
    int px,py,xspeed,yspeed;
    //标志图片序列,每帧图宽、高,当前帧
    int pr,pc,pw,ph,cf;
    GameSurfaceView gsv;
    Direction dir = Direction.DOWN;
    …
    public void draw(Canvas canvas,Paint paint){
        move();
        canvas.save();
        //canvas.translate(px, py);
        //计算当前帧的坐标
        int x  = pw * (cf % 4);
        int y = ph * pr;
        canvas.clipRect(px,py, px + pw, py + ph);
        canvas.drawBitmap(lhch, px - x,py - y, paint);
        Log.i("S","当前行、帧 : " + pr + "," + cf % 4);
```

```
            cf++;
            canvas.restore();
        }
        …
```

该项目的核心是确定所播放图片的区域，然后在此区域实现剪切区即可。代码07-27的详细解释如下。

(1) x=pw*(cf%4)计算当前帧相对于整个位图的 x 坐标，其中 pw 是通过计算所得(pw=lhch.getWidth()/4，整个位片的宽度除以4)每一帧图的宽度，cf 是当前帧所处在的列(请查看图7.16，人物的每个方向的行走都与列相关)。

(2) y=ph*pr计算当前帧相对于整体位图的 y 坐标，其中 ph 是每帧图的高度(ph=lhch.getHeight()/4)，pr 是当前行，可以根据人物移动方向确定。

(3) canvas.clipRect(px,py, px+pw, py+ph)确定剪切区，大小与每帧图宽高一致。

(4) canvas.drawBitmap(lhch, px-x,py-y, paint)绘制位图，因为绘制时还是整张位图都绘制，所以整个位图的坐标要移动 $-x$ 和 $-y$ 大小。

7.6 游戏元素的碰撞

所谓游戏元素的碰撞就是检测两个游戏元素有无位置上的重叠，通过每个元素的坐标和宽度可以确定一个区域，如果两个元素有重叠区域就视为碰撞。

7.6.1 矩形碰撞

矩形碰撞是通过检测矩形的坐标、宽高有无重叠来确定的，常用于检测两个图片之间的位置关系。两个矩形的位置无非碰撞或不碰撞，如果要检测碰撞，则需要写一个很复杂的判断条件，相反地，如果要检测二者不碰撞，判断条件相对简单得多，所以矩形的碰撞是通过排除不碰撞的条件来确定的。

对于如图7.17所示的两个矩形不发生碰撞的条件如下。

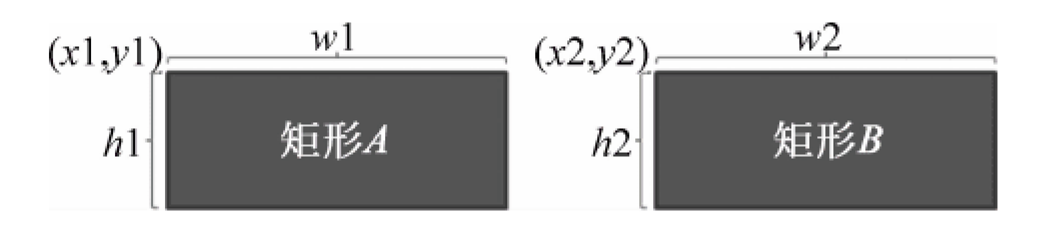


图7.17 矩形位置关系

(1) 矩形 A 在矩形 B 的左边：x1<x2&&x1+w1<x2。

(2) 矩形 A 在矩形 B 的右边：x1>x2&&x1>x2+w2。

(3) 矩形 A 在矩形 B 的上边：y1<y2&&y1+h1<y2。

(4) 矩形 A 在矩形 B 的下边：y1>y2&&y1>y2+h2。

除了上述条件外，矩形 A 与矩形 B 有交集，发生碰撞。上述判断方法也适用于图形与矩形的碰撞检测。

7.6.2 圆形碰撞

圆形是通过圆心和半径确定的，圆形的碰撞则是通过检验圆心距与半径之和的关系（数学与物理中的知识将被广泛用于游戏开发中）。与判断矩形关系不同，可以通过圆心距公式（$d=\sqrt{(x1-x2)^2+(y1-y2)^2}$）直接判断两圆关系。两圆心距公式反映到代码中是 Math. sqrt(Math. pow(x1－x2,2)＋Math. pow(y1－y2,2))。

7.6.3 Region

Region 类在前面已经有所介绍，它还有一个方法 contains(int x, int y)，可以检验所指定的坐标(x,y)是否位于 Region 区域中。可以使用 Region 检验某个区域是否被单击。

新建项目 part07_13，演示上述元素碰撞，完整代码请参考随书配套资料。MyRect. java 是自定义矩形，关键功能如代码 07-28 所示；MyCircle. java 是自定义圆形，关键功能如代码 07-29 所示，运行效果如图 7. 18 所示。

代码 07-28

```
…
//移动到指定位置
public void drawTo(int x, int y) {
    this.px = x - this.pw / 2;
    this.py = y - this.ph / 2;
}
//判断是否单击
public boolean isClick(int x,int y){
    // 如果单击了该矩形区域
    region = new Region(px, py, px + pw, py + ph);
    if(region.contains(x, y)){
        isclick = true;
        return true;
    }else
        isclick = false;
        return false;
}
//判断是否碰撞
public boolean isCollide(MyRect mr){
    if(this.px < mr.px&&this.px + this.pw < mr.px)return false;
    if(this.px > mr.px&&this.px > mr.px + mr.pw)return false;
    if(this.py < mr.py&&this.py + this.ph < mr.py)return false;
    if(this.py > mr.py&&this.py > mr.py + mr.ph)return false;
    return true;
}
…
```

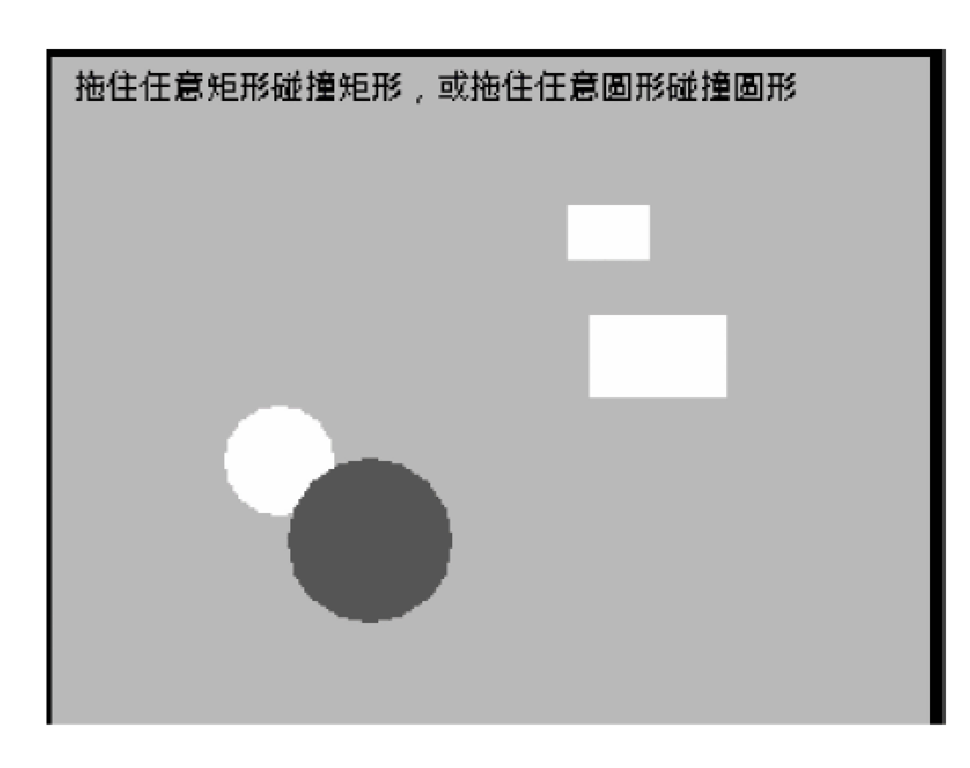


图 7.18　元素碰撞效果

代码 07-29

```
…
//拖动到指定位置
public void drawTo(int x, int y) {
    this.px = x;
    this.py = y;
}
//判断是否单击了,被单击了将会移动
public boolean isClick(int x,int y){
    // 如果单击了该矩形区域,点到圆心的距离与半径比较
    int d = (int) Math.sqrt(Math.pow(px - x, 2) + Math.pow(py - y, 2));
    if(d < pr){
        isclick = true;
        return true;
    }else
        isclick = false;
        return false;
}
//判断是否碰撞
public boolean isCollide(MyCircle mc){
    int d = (int) Math.sqrt(Math.pow(this.px - mc.px, 2) + Math.pow(this.py - mc.py, 2));
    if(d < this.pr + mc.pr)
        return true;
    else
        return false;
}
…
```

习　　题

1. 如何实现横竖屏切换？
2. 如何实现全屏？
3. Region 的运算有哪些？简要说明它们的作用。
4. 如何实现帧动画？
5. 请完善 7.5.3 节中的游戏功能。



CHAPTER 8

音频与视频的使用

本章主要内容：

声音与视频的操作；
MediaPlayer；
SoundPool；
VideoView。

Android 提供了非常丰富的多媒体操作 API，使得实现音频与视频播放的操作变得比较简单。Android 支持的音频格式有 mp3、wav 和 3gp 等，支持的视频格式有 mp4 和 3gp 等。Android 系统所支持的全部音频与视频格式可以通过（sdk 路径）/docs/guide/appendix/media-formats.html 查看。

8.1 MediaPlayer

MediaPlayer 类位于 android.media 包中，是 Android 系统中重要的多媒体操作类，可以用于播放音频文件，也可以播放视频文件。

8.1.1 创建 MediaPlayer

MediaPlayer 可以通过构造方法直接创建，也可以调用静态方法 create 创建，详见表 8.1。

表 8.1 创建 MediaPlayer 的常用方式

方法声明	说明
MediaPlayer()	构造方法
static MediaPlayer create(Context context, int resid)	通过指定音频资源创建播放器，音频资源文件可以放在 res/raw 文件夹下
static MediaPlayer create(Context context, Uri uri)	通过 uri 创建播放器

音频文件可以放在项目资源 res/raw 文件夹中，如果 raw 文件夹不存在，可以直接创

建，该文件夹下只能存放 Android 支持的文件格式，文件名符合标识符命名规则。注意，该文件夹中的文件，不会被编译成二进制，而是直接打包到 apk 文件中。

8.1.2　设置播放文件

MediaPlayer 中的播放文件可以在创建时设定，这种方式可以直接调用 start 方法进行播放；还可以通过 setDataSource 方法设置，这种方法需要先调用 prepare 方法，然后再调用 start 方法播放。常用的设置方法如下所示。

（1）setDataSource（String path），播放指定路径的文件，可用于播放 SD 卡上的音频资源。

（2）setDataSource(Context context, Uri uri)，播放指定 uri 的资源，该 uri 应该是可以被下载的。

新建项目 part08_1，创建播放器，播放指定的网络资源，完整代码请查看随书配套资料，如代码 08-1 所示。

代码 08-1

```
public class MainActivity extends Activity {
    MediaPlayer player;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //初始化播放器
        player = new MediaPlayer();
        //创建资源
        Uri uri = Uri.parse("http://xxxxxxxx.mp3");
        try {
            //加载资源
            player.setDataSource(this, uri);
            //准备
            player.prepare();
            Log.i("S", "准备完成");
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SecurityException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalStateException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        findViewById(R.id.button1).setOnClickListener(new OnClickListener(){
```

```
                @Override
                public void onClick(View v) {
                    //播放器播放
                    player.start();
                }
            });
        }
        @Override
        protected void onPause() {
            super.onPause();
            if(player!= null){
                player.release();
                player = null;
            }
        }
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is present.
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
        }
    }
```

运行程序，单击按钮即可开始播放指定网络音频。注意，该项目需要联网，应该在配置文件 Manifest. xml 中开启网络权限 < uses-permission android: name = " android. permission. INTERNET"/>。

8.1.3 播放器的控制

MediaPlayer 提供了丰富的方法用于播放的控制，详见表 8.2（播放器的例子可以参考第 6 章代码 06-6）。

表 8.2 MediaPlayer 常用的控制方法

返回值类型	方法声明	说明
int	getCurrentPosition()	获取当前的播放位置
int	getDuration()	获取音视频总时长
int	getVideoHeight()	获取视频高度
int	getVideoWidth()	获取视频宽度
boolean	isLooping()	检测播放器是否处于循环播放状态
boolean	isPlaying()	检测播放器是否处于播放状态
void	pause()	暂停操作
void	prepare()	准备操作，通过 create 加载的文件无法调用该方法
void	prepareAsync()	异步准备操作
void	release()	释放资源，退出时需要调用该方法
void	reset()	可以让播放器从错误状态恢复
void	seekTo(int msec)	播放位置滑动到指定时间，单位毫秒

续表

返回值类型	方 法 声 明	说　　明
void	setLooping(boolean looping)	设置是否循环
void	setVolume (float leftVolume, float rightVolume)	设置左右声道的音量
void	start()	开始或继续播放
void	stop()	停止播放

8.1.4　播放器的监听器

MediaPlayer 提供了一些设置不同监听器的方法来更好地对播放器的工作状态进行监听,可以通过这些监听器,对播放过程中的事件进行处理。常用的播放器如下。

(1) setOnCompletionListener(MediaPlayer. OnCompletionListener listener),当播放完成时触发监听器。

(2) setOnErrorListener(MediaPlayer. OnErrorListener listener),监听错误发生。

(3) setOnPreparedListener(MediaPlayer. OnPreparedListener listener),监听播放资源准备就绪。

(4) setOnSeekCompleteListener(MediaPlayer. OnSeekCompleteListener listener),监听 seek 方法执行。

(5) setOnVideoSizeChangedListener (MediaPlayer. OnVideoSizeChangedListener listener),监听视频变化。

8.2　SoundPool

MediaPlayer 在播放音乐文件时资源占用比较多,有一定的延时,不支持多个音频同时播放,但是可以播放较大的音频文件,所以常用于播放器的开发、游戏背景音乐的播放等场合。SoundPool 也可以播放音频文件,支持同时播放多个音频,但最大只能申请 1MB 内存空间,仅能用于播放很短的声音片段,所以 SoundPool 常用于播放游戏或软件中的声音特效。

SoundPool 主要用于播放声音片段,CPU 占用低,延时较小,可同时播放多个声音。SoundPool 类位于 android. media 包中,可以通过构造方法创建 SoundPool (int maxStreams, int streamType, int srcQuality)。

(1) maxStreams,最大支持的音频数量。

(2) streamType,音频类型,可以设置为 AudioManager. STREAM_MUSIC。

(3) srcQuality,声音品质,默认为 0。

创建 SoundPool 对象后,可以通过 load 方法,向池中预先加载音频文件。常用的 load 方法描述如下。

(1) load(Context context, int resId, int priority),加载资源包中的音频文件,resId 是 raw 文件夹中音频文件对于 R 中的引用。

(2) load(String path, int priority),通过路径加载音频文件。

上述两个方法中的priority参数是优先级,当前没有启用(ADK 4.2文档这样解释),建议设置为1,便于向后兼容。load方法会返回int类型的数值(该数值如同池中音频对应的ID),该返回值需要保存,后面播放池中的音频文件时,需要指定该数值。SoundPool使用方法play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)播放指定音频,其参数含义如下。

(1) soundID,音频的ID,该值是在load方法执行时的返回值。

(2) leftVolume、rightVolume,左右音量,取值范围是0.0～1.0。

(3) priority,优先级,数值越大优先级越高,最低是0。

(4) loop,循环次数,取值0为不循环,－1为循环。

(5) rate,播放速率,取值范围是0.5～2.0,正常播放速率为1。

以上介绍的顺序即为SoundPool播放声音的步骤。新建项目part08_2,演示SoundPool的使用。布局文件中含有三个按钮,单击时播放对应音频。项目运行前,需要在res资源中新建raw文件夹,并添加相应音频资源,可以使用随书配套资料中的音频文件。MainActivity.java原有文件参考代码08-2。

代码 08-2

```
public class MainActivity extends Activity implements OnClickListener {
    SoundPool sp;
    Map<String, Integer> sm = new HashMap<String, Integer>(); //保存load后的返回值 ①
    Button b1,b2,b3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1 = (Button) findViewById(R.id.button1);
        b2 = (Button) findViewById(R.id.button2);
        b3 = (Button) findViewById(R.id.button3);
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
        b3.setOnClickListener(this);
        //创建SoundPool
        sp = new SoundPool(5,AudioManager.STREAM_MUSIC,5);
        //加载音效文件
        sm.put("1",sp.load(this, R.raw.dummy_break_05, 1));     ①
        sm.put("2",sp.load(this, R.raw.footstep_water_01, 1));
        sm.put("3",sp.load(this, R.raw.woman_scream, 1));
    }
    @Override
    public void onClick(View arg0) {
        //单击按钮播放不同音效
        switch(arg0.getId()){
        case R.id.button1:
            sp.play(sm.get("1"), 1.0f, 1.0f, 1, 0, 1.0f);     ②
            break;
```

```
            case R.id.button2:
                sp.play(sm.get("2"), 1.0f, 1.0f, 1, 0, 1.0f);
                break;
            case R.id.button3:
                sp.play(sm.get("3"), 1.0f, 1.0f, 1, 0, 1.0f);
                break;
            }
        }
    }
```

在使用 SoundPool 时注意：①加载音频文件时一定要保存返回值；②播放时指定相应的音频 ID，该 ID 不是资源文件在 R 中生成的 ID，而是 load 时的返回值。

对于 MediaPlayer 和 SoundPool 的使用场景应该注意以下几点。

- MediaPlayer 支持播放较大的音频文件，占用资源较大，常用于开发播放器。
- SoundPool 支持同时播放多个音频文件，但申请内存空间有限制，可用于播放声音片段。
- SoundPool 播放的音频文件建议使用 ogg 格式音频。

8.3 VideoView

VideoView 是 Android 提供的系统控件，用于播放视频。VideoView 类位于 android.widget 包中，使用方法与 MediaPlayer 较为类似。VideoView 提供以下两个方法用于加载视频文件。

(1) setVideoPath(String path)，指定文件路径加载视频，用于播放 SD 卡中的视频。

(2) setVideoURI(Uri uri)，指定 uri 加载视频，用于播放网络中的视频。

新建项目 part08_3，演示 VideoView 的用法。MainActivity.java 源文件如代码 08-3 所示，布局文件 activity_main.xml 比较简单，添加一个 VideoView 控件，两个 ImageButton 用于控制视频的播放。

代码 08-3

```
public class MainActivity extends Activity implements OnClickListener {
    VideoView vv;
    ImageButton ib1, ib2;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        vv = (VideoView) findViewById(R.id.videoView1);
        File file = new File("/mnt/sdcard/xy.3gp");
        if(file.exists()){
            vv.setVideoPath(file.getAbsolutePath());
        }
        ib1 = (ImageButton) findViewById(R.id.ib_play);
        ib2 = (ImageButton) findViewById(R.id.ib_stop);
        ib1.setOnClickListener(this);
```

```
            ib2.setOnClickListener(this);
        }
        public void onClick(View v) {
            if(v.getId() == R.id.ib_play){
                vv.start();
            }else{
                vv.pause();
            }   }}
```

项目运行效果如图 8.1 所示，可以通过两个按钮控制视频的播放和暂停。需要注意的是，项目运行时虚拟机的 SD 卡中应该先放入所需视频（Eclipse 切换到 DDMS 视图，通过 push 按钮向虚拟机中复制文件），其格式为 mp4 或 3gp，且视频质量不应该太高，否则运行时模拟器很不流畅，甚至只显示黑色。

VideoView 控件的控制操作也可以由 MediaController 完成，该类位于 android.widget 包中。MediaController 带有“播放”、“暂停”、“上一个”、“下一个”等按钮，使用这些按钮就可以完成对 VideoView 的控制，不过在使用之前需要将 VideoView 和 MediaController 建立连接。

代码 08-4 演示 MediaController 与 VideoView 的使用，运行效果如图 8.2 所示。布局文件非常简单，只需要一个 VideoView 控件即可。MediaController 与 VideoView 建立连接点代码详见代码 08-4 中的①。

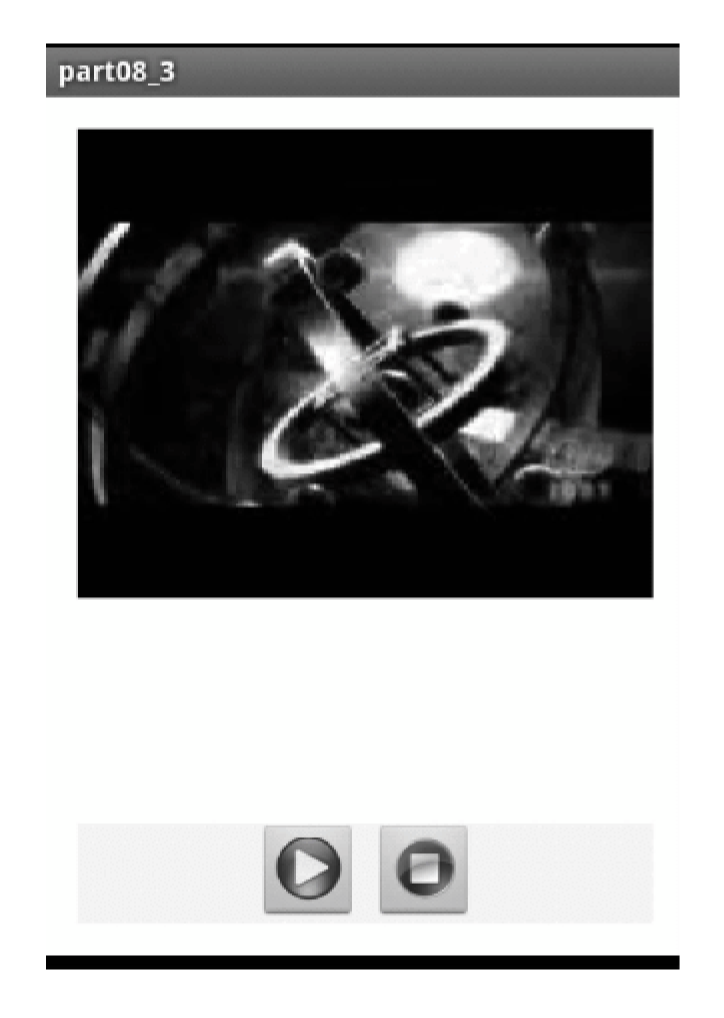


图 8.1　VideoView 播放效果图

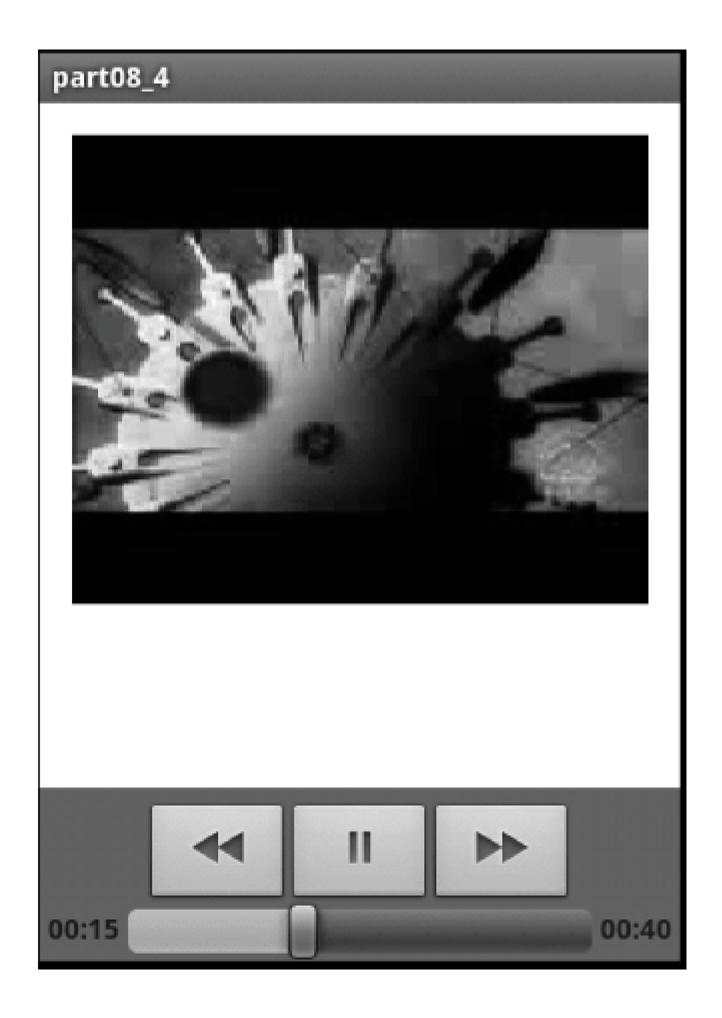


图 8.2　MediaController 控制播放效果

代码 08-4

```
public class MainActivity extends Activity{
    VideoView vv;          //视频播放控件
    MediaController mc;   //播放控制器
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
vv = (VideoView) findViewById(R.id.videoView1);
//创建 MediaController
mc = new MediaController(this);
File file  = new File("/mnt/sdcard/xy.3gp");
if(file.exists()){
    vv.setVideoPath(file.getAbsolutePath());
    //播放器和控制建立连接
    vv.setMediaController(mc);  ①
    mc.setMediaPlayer(vv);
    vv.requestFocus();
}   }}
```

8.4　MediaRecoder

MediaRecoder 位于 android.media 包中，用于录制声音和视频。MediaRecoder 状态的改变、调用方法的先后顺序都可以从图 8.3 得到。

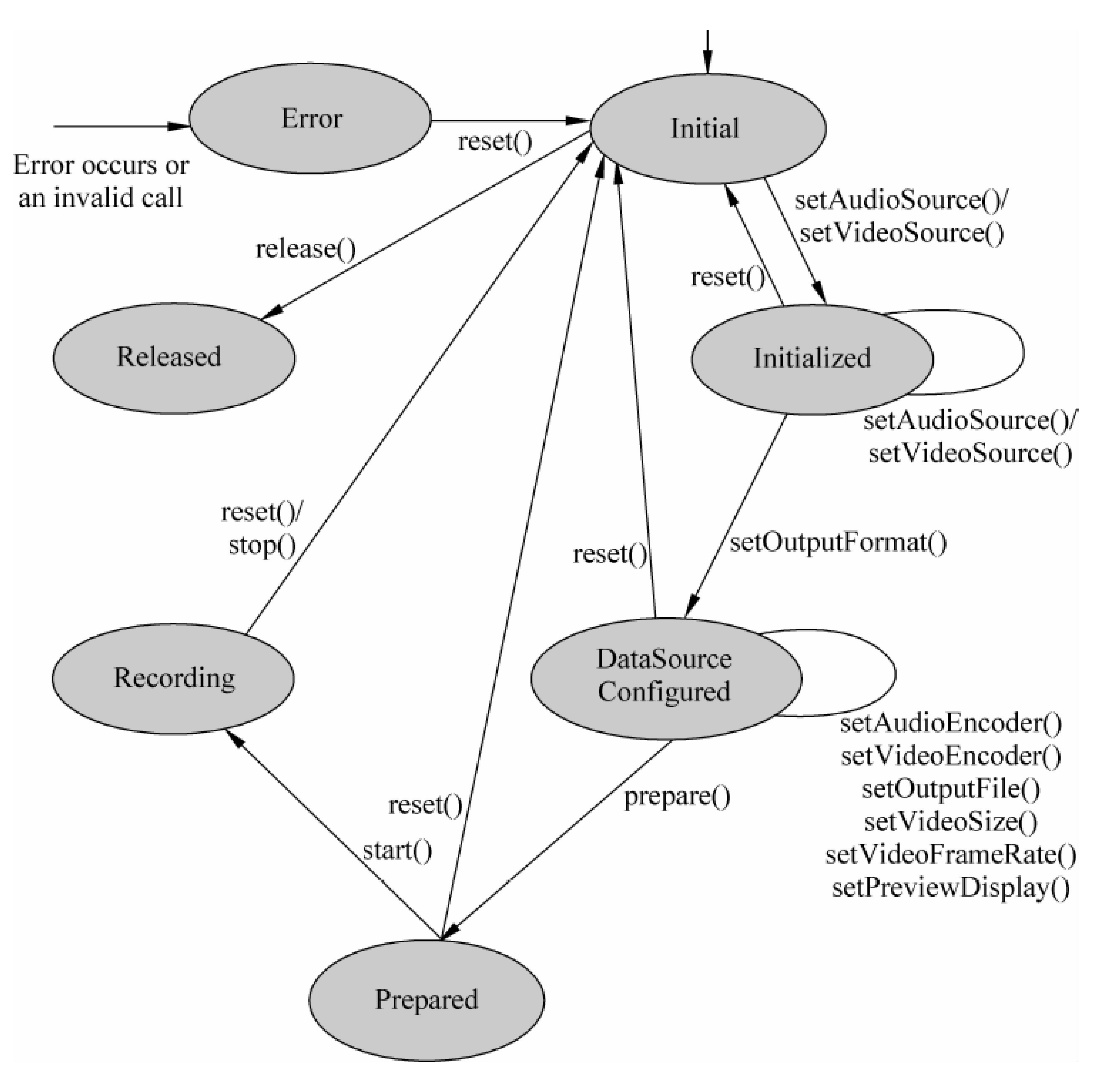


图 8.3　MediaRecoder 状态转换图

MediaRecoder 常用方法的说明详见表 8.3，其中线框加粗的单元格中的方法是录制音频和视频都需要设置的方法。

表 8.3 MediaRecoder 常用方法说明

方法声明	说　明
setAudioSource(int audio_source)	设置音频来源，其值可以调用内部类 MediaRecorder.AudioSource 的静态值，MIC 表示来源于手机麦克
setOutputFormat(int output_format)	设置输出格式，其值可以调用内部类 MediaRecorder.OutputFormat 的静态值：AMR_NB、AMR_WB、DEFAULT、MPEG_4、RAW_AMR、THREE_GPP
setAudioEncoder(int audio_encoder)	设置音频的编码格式，值为内部类 MediaRecorder.AudioEncoder 的静态值：AAC、AMR_NB、AMR_WB、DEFAULT
setAudioEncodingBitRate (int bitRate)	设置音频的编码率，可以不设置，采用默认值
setAudioSamplingRate (int sRate)	设置音频采样率，可以不设置，采用默认值
setOutputFile(String path)	录制文件的保存位置
prepare()	准备录制
start()	开始录制
stop()	停止录制
release()	释放资源
setVideoSource(int video_source)	设置视频来源，其值采用内部类 MediaRecorder.VideoSource 的静态值，CAMERA 表示来源于摄像头
setVideoEncoder (int video_encoder)	设置视频的编码格式，值为内部类 MediaRecorder.VideoEncoder 的静态值：DEFAULT、H263、H264、MPEG_4_SP
setVideoEncodingBitRate(int bitRate)	设置视频编码的比特率
setVideoSize(int width, int height)	设置视频的大小尺寸，必须设置
setVideoFrameRate (int rate)	设置帧速率，设置完视频来源后，必须设置该方法
setPreviewDisplay (Surface sv)	设置预览视频的界面

8.4.1 录制声音

Android 录制声音的过程比较固定，按照代码 08-5 中的 6 个步骤操作即可。完整的项目代码请查阅随书配套资料 part08_5 项目。

代码 08-5

```
//录音
public void recorde(){
    //录音文件需要放置在 SD 卡
    //MEDIA_MOUNTED 状态为 SD 卡正常使用状态
    if(!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
        Toast.makeText(this, "SD 卡无法正常使用，录音结束!", Toast.LENGTH_LONG).show();
        return;
    }
    try {
        //SD 卡根路径
        File sdroot = Environment.getExternalStorageDirectory();
        //保存录音文件
```

```
            File file = new File(sdroot,System.currentTimeMillis() + ".amr");
            //1.创建录音器
            mrecorder = new MediaRecorder();
            //2.设置音频来源
            mrecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            //3.设置输出格式
            mrecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            //4.设置编码格式
            mrecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
            //5.设置输出音频路径
            mrecorder.setOutputFile(file.getAbsolutePath());
            //6.准备开始
            mrecorder.prepare();
            mrecorder.start();
        } catch (IllegalStateException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

项目运行后，单击“录制”按钮便可在 SD 卡上创建录音文件。对于 SD 卡的读写将在第 9 章中详细介绍。录音结束后需要释放 MediaRecorder 所占用的资源。注意开启相应的权限：

(1) <uses-permission android:name="android.permission.RECORD_AUDIO"/>，录制声音权限。

(2) <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>，写 SD 卡权限。

8.4.2 录制视频

MediaRecorder 用于录制视频的过程与录制声音基本一致，具体的先后顺序如图 8.3 所示。不同于单独录制音频的是，在相应步骤要设置视频的录制格式、编码等，核心功能参考代码 08-6，完整代码请查阅随书配套资料项目 part08_6。运行代码前需要添加相应权限：

(1) <uses-permission android:name="android.permission.RECORD_AUDIO"/>，启用录制声音权限。

(2) <uses-permission android:name="android.permission.CAMERA"/>，打开摄像机权限。

(3) <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>，启用写入 SD 卡权限。

代码 08-6

```
//录制视频
public void recorde(){
    //录音文件需要放置在 SD 卡
```

```
        //MEDIA_MOUNTED 状态为 SD 卡正常使用状态
        if(!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
            Toast.makeText(this, "SD 卡无法正常使用,录制结束!", Toast.LENGTH_LONG).show();
            return;
        }
        try {
            //SD 卡根路径
            File sdroot = Environment.getExternalStorageDirectory();
            //保存录音文件
            File file  = new File(sdroot,System.currentTimeMillis() + ".3gp");
            //1.创建录音器
            mrecorder  = new MediaRecorder();
            //2.设置音频、视频来源
            mrecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
            mrecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            //3.设置输出格式
            mrecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            //4.设置音频、视频编码格式
            mrecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);
            mrecorder.setVideoSize(320, 240);
            //mrecorder.setVideoFrameRate(15);
            mrecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
            //5.设置输出路径
            mrecorder.setOutputFile(file.getAbsolutePath());
            //6.视频预览设置
            mrecorder.setPreviewDisplay(sv.getHolder().getSurface());
            //7.准备录制
            mrecorder.prepare();
            mrecorder.start();
        } catch (IllegalStateException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

该项目布局文件比较简单,含有三个控件(如代码 08-7 所示),两个按钮分别用于录制和停止;一个 SurfaceView 控件用于预览录制的视频。

该项目在虚拟机上运行没有效果,原因在于虚拟机没有摄像头的硬件支持。在真机上的运行效果如图 8.4 所示。

代码 08-7

```
<RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
```

```
    tools:context = ".MainActivity" >
   <SurfaceView
        android:id = "@ + id/sv"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent"
        />
    <LinearLayout
        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:orientation = "horizontal"
        android:gravity = "center_horizontal"
        android:layout_alignParentBottom = "true"
        >
        <Button
            android:id = "@ + id/ib_recorde"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "录制"
            />
        <Button
            android:id = "@ + id/ib_stop"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "停止"
            />
    </LinearLayout>
</RelativeLayout>
```

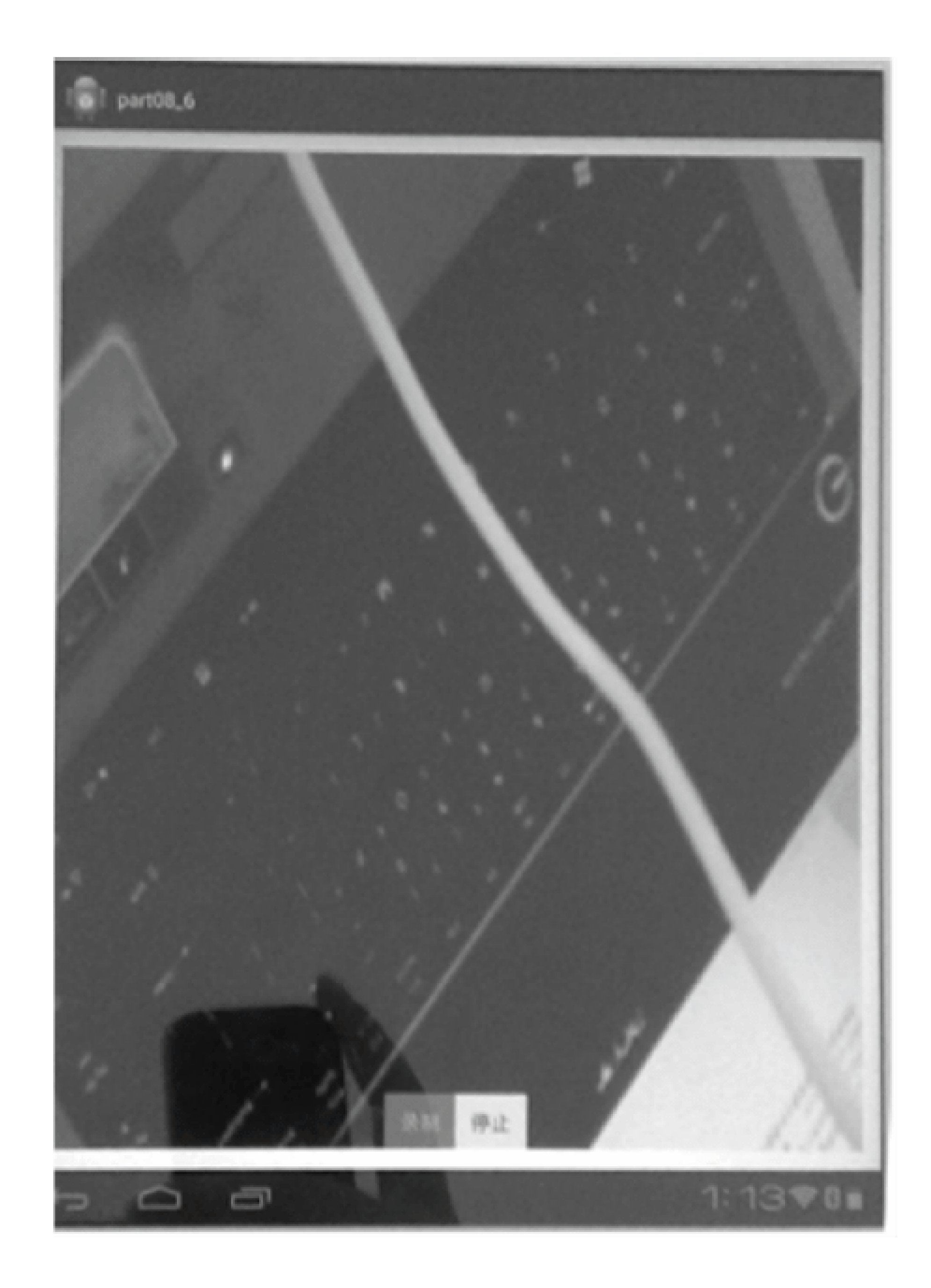


图 8.4　真机视频录制效果图

使用 MediaRecoder 记录声音、视频时需要注意以下几点。

- MediaRecoder 方法的执行要遵循如图 8.3 所示的先后顺序，比如在执行 setOutputFormat()方法之后再执行 setAudioEncoder()等方法。
- 录制声音和视频需要麦克和摄像头，不要忘记开启相应权限。

习 题

1. MediaPlayer 和 SoundPool 都可以播放声音，它们有何异同？各适用于何种场合？
2. 请简要说明 SoundPool 类的方法 play 参数的意义是什么？
3. 完善 8.3 节视频播放器的基本功能，使其可以播放 SD 卡中的视频文件。
4. MediaRecorder 对象的状态有哪些？在状态发送改变时需要注意什么？
5. 将 8.3 节视频播放功能和 8.4.2 节视频录制功能进行有机融合，实现录制视频并播放的功能。



CHAPTER 9

第9章

数据的存储

本章主要内容：

Android 中数据存储的三种方式；
SharedPreferences 与 Editor；
I/O 流；
读写 SD 卡；
嵌入式数据库 SQLite；
SQLiteDatabase 类；
SQLiteOpenHelper 类。

本章主要介绍 Android 系统三种数据保存的方式，分别是使用 SharedPreferences 类进行文件的读写；使用 I/O 的相关知识进行文件的读写；使用 SQLiteDatabase 数据库读写数据。

9.1 SharedPreferences 读写 XML 文件

SharedPreferences 类位于 android.content 包中，是一个接口。借助 SharedPreferences 只能完成读取文件的操作，如果要写入文件还需要用到 SharedPreferences 的内部类 Editor。二者操作的文件都是 XML 文件，此类文件可以用于存储较为简单的字符串类型数据，如应用程序的配置信息，是否开启声音、震动，游戏人物的等级、经验等数据。

9.1.1 SharedPreferences 基本操作

由于 SharedPreferences 只是一个接口，而且 Android 没有提供相关的实现类，所以无法直接创建 SharedPreferences 的实例，需要通过 Context 类（Activity 是其间距子类）的 getSharedPreferences (String name, int mode)方法获取 SharedPreferences 类的实例。其中，参数 name 是文件的名称，参数 mode 的取值及含义如下。

(1) MODE_PRIVATE，声明 SharedPreferences 操作的文件只能供本应用程序使用，其他应用程序无法访问。

(2) MODE_WORLD_READABLE,声明 SharedPreferences 操作的文件可以被其他应用程序读取,但不能写入。

(3) MODE_WORLD_WRITEABLE,声明 SharedPreferences 操作的文件可以被其他应用程序读写。

(4) MODE_MULTI_PROCESS,新版的 ADK 中已经不再使用该属性。

获取 SharedPreferences 实例后,可以使用相应的方法读取参数 name 所指定的文件。SharedPreferences 操作的文件是 XML 格式的,无论读还是写都是以键值对的形式进行,与 Map 的操作类似。常用的操作如表 9.1 所示。

表 9.1 SharedPreferences 类常用的操作

返回值	声　明	说　明
abstract boolean	contains(String key)	检验是否包含 key 键
SharedPreferences. Editor	edit()	获取 Editor 对象
Map<String, ? >	getAll()	获取所有键值对
boolean	getBoolean (String key, boolean defValue)	获取指定 key 对应的值,如果不存在,则返回默认值 defValue
float	getFloat(String key, float defValue)	同上
int	getInt(String key, int defValue)	同上
long	getLong(String key, long defValue)	同上
String	getString(String key, String defValue)	同上
Set<String>	getStringSet(String key, Set<String> defValues)	获取 key 对应的 Set 集

9.1.2 Editor 写入数据

Editor 也是一个接口,只能通过 SharedPreferences 类的 edit()方法获取实例。通过 SharedPreferences 与 Editor 可以完成对指定文件的读写操作(注意该文件隶属于本应用程序)。Editor 类常用的方法如表 9.2 所示。

表 9.2 Editor 类常用的方法

返回值	声　明	说　明
SharedPreferences. Editor	clear()	清除所有数据
boolean	commit()	提交数据,Editor 中的数据只有在该方法执行之后才会存入文件
SharedPreferences. Editor	putBoolean(String key, boolean value)	以 key 为键,保存 value
SharedPreferences. Editor	putFloat(String key, float value)	同上
SharedPreferences. Editor	putInt(String key, int value)	同上
SharedPreferences. Editor	putLong(String key, long value)	同上
SharedPreferences. Editor	putString(String key, String value)	同上
SharedPreferences. Editor	putStringSet(String key, Set<String> values)	以 key 为键,保存 set 集合
SharedPreferences. Editor	remove(String key)	移除 key 所对应的数值

下面演示 SharedPreferences 与 Editor 的应用。新建项目 part09_1，MainActivity.java 的代码如代码 09-1 所示，布局文件为 activity_main.xml，详见随书配套资料。

📖代码 09-1

```
public class MainActivity extends Activity implements OnClickListener{
    EditText et1,et2;
    Button b1,b2;
    SharedPreferences sf;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et1 = (EditText) findViewById(R.id.et_name);
        et2 = (EditText) findViewById(R.id.et_pwd);
        b1 = (Button) findViewById(R.id.bt_save);
        b2 = (Button) findViewById(R.id.bt_read);
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
        //获取 SharedPreferences 实例,读写 login.xml 文件,方式为独占
        sf = getSharedPreferences("login",Context.MODE_PRIVATE);
    }
    @Override
    public void onClick(View view) {
        switch(view.getId()){
        case R.id.bt_save:
            save();         break;
        case R.id.bt_read:
            read();         break;
        }
    }
    //读取文件
    private void read() {
        //读取字符串
        String name = sf.getString("name", "请输入新登录名");
        String pwd = sf.getString("pwd", "");
        et1.setText(name);
        et2.setText(pwd);
    }
    //保存文件
    private void save() {
        //获取 Editor 实例
        Editor et = sf.edit();
        //保存字符串
        et.putString("name", et1.getEditableText().toString());
        et.putString("pwd", et2.getEditableText().toString());
        et.commit();
        et1.setText("");
        et2.setText("");
    }
}
```

项目运行效果如图 9.1 所示。将 Eclipse 切换为 DDMS 试图(选择 Window→Open

Perspective→Other→DDMS 命令)，打开 File explorer 选项卡，依次打开文件夹 data→data→com. freshen. code(此处为应用程序包名)→shared_prefs，程序中新建的文件 login. xml 就位于此处。将该文件从虚拟机中复制出(通过右上角的 pull 按键)，打开文件显示内容如代码 09-2 所示。



图 9.1 保存数据运行效果

SharedPreferences 保存的数据文件为 XML 格式，根标签为 map，以键值对的形式存放数据。所以 SharedPreferences 常作为轻量级数据存储实用，提供了 Android 平台常规的 long 长整型、int 整型、String 字符串等保存类型。

代码 09-2

```
<?xml version = '1.0' encoding = 'utf - 8' standalone = 'yes' ?>
< map >
< string name = "pwd"> admin </string >
< string name = "name"> admin </string >
</map >
```

9.2 使用 I/O 读写文件

Android 兼容 J2SE 中 I/O 的操作(InputStream、OutputStream、Reader、Writer)，因此可以借助 I/O 的知识读写应用程序或 SD 卡中的数据。

9.2.1 读写应用程序中的文件

保存数据的文件可以放在应用程序中，这些文件将来是位于手机内存中的，也可以选择放在 SD 卡上。一般而言，放在手机内存的文件都比较小，且经常读写；SD 卡则可以保存较大数据文件，且这些数据的丢失不影响应用程序的正常运行。

与获取 SharedPreferences 实例类似，Context 类提供了以下两个方法，分别用于获取文件读取流和文件写入流。

(1) abstract FileInputStream openFileInput(String name)，打开文件获取读取数据流。

(2) abstract FileOutputStream openFileOutput(String name, int mode)，向文件中写入数据，写入的方式由 mode 决定。Mode 的取值为：MODE_PRIVATE 应用程序独占文件，不允许其他应用程序读写；MODE_ APPEND 向已有文件中追加内容；MODE_

WORLD_READABLE 其他应用程序可以读取；MODE_WORLD_WRITEABLE 其他应用程序可以读写。

新建项目 part09_2，演示应用程序文件读写方式。布局文件 activity_main.xml 比较简单，详细代码可以查阅随书配套资料。MainActivity.java 中读写数据的方法如代码 09-3 所示。

代码 09-3

```
//读取文件
private void read() {
    BufferedReader br = null;
    FileInputStream fis = null;
    try {
        tv1.setText("");
        fis = openFileInput("content.txt");
        br = new BufferedReader(new InputStreamReader(fis));
        String ln = null;
        while( (ln = br.readLine())!= null ){
            tv1.append(ln + "\n");
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }finally{
        if(br!= null)
            try {
                br.close();
            } catch (IOException e) {
                // TODO Auto - generated catch block
                e.printStackTrace();
            }
    }
}
//保存文件
private void save() {
    if(et1.getText().toString().length()< 1||et2.getText().toString().length()< 1)return;
    FileOutputStream fos = null;
    PrintWriter pw = null;
    try {
        //追加方式打开文件 content.txt
        fos = openFileOutput("content.txt", Context.MODE_APPEND);
        pw  = new PrintWriter(fos);
        pw.println("好友姓名: " + et1.getText().toString() + "\t 电话号码:" + et2.getText
().toString());
        pw.flush();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }finally{
        if(pw!= null)pw.close();
    }
}
```

项目运行效果如图 9.2 所示。从代码 09-3 可以看出，Android 中对文件的操作基本上与 J2SE 中 I/O 的知识一致。具有 Android 系统自身特点的就是方法 openFileInput(String name)和 openFileOutput(String name, int mode)，这两个方法是 Context 提供的(Activity 间接继承 Context)，用于获取指定文件的读取流和写入流。



图 9.2　文件读写运行效果图

那么指定文件的位置在哪里呢？图 9.3 是在 DDMS 视图下截取的文件目录结构图(通过 data→data 就可以找到)，该目录是本应用程序安装后对应的目录，其中 shared_prefs 文件夹是用于存储通过 SharedPreferences 保存的文件，files 则是上述两个方法保存的文件，lib 用于存储相关类库文件。需要注意的是，这些文件夹都存在于手机内存中。

com.freshen.code		2013-08-30	03:13	drwxr-x--x
files		2013-08-30	03:14	drwxrwx--x
content.txt	89	2013-08-30	03:14	-rw-rw----
lib		2013-08-30	03:13	drwxr-xr-x
shared_prefs		2013-08-29	10:19	drwxrwx--x
login.xml	140	2013-08-29	10:19	-rw-rw----

图 9.3　应用程序目录结构图

9.2.2　读写 SD 卡中的文件

对于较大文件或需要下载保存的文件，建议将其保存在手机的 SD 卡上，本节介绍手机 SD 卡的读写。由于手机 SD 卡对于手机来说不是必需的，在读写前需要检验手机 SD 卡的状态，如是否存在 SD 卡，是否可以读写等。Android 提供了相关的 API 用于操作手机 SD 卡。

Environment 类位于 android.os 包中，是用于检验和获取手机 SD 卡(或称为外存储设备)状态、根路径等信息的工具类。Environment 类中的方法都是静态方法，可以直接调用，常用的方法详见表 9.3。

表 9.3　Environment 类常用的方法

返回值类型	方　法	说　明
File	getExternalStorageDirectory()	获取外存储设备的根路径，一般情况下是/mnt/sdcard
String	getExternalStorageState()	获取外存储设备的状态，具体状态及含义说明详见表 9.4

续表

返回值类型	方法声明	说明
File	getRootDirectory()	获取 Android 系统根路径
boolean	isExternalStorageEmulated()	判断外部存储是否有效、可用
boolean	isExternalStorageRemovable()	判断外部存储是否可以移除

表 9.4 SD 卡状态及说明

SD 卡状态值	含义	SD 卡是否可读写
MEDIA_MOUNTED	SD 卡正常挂载	可以
MEDIA_REMOVED	无介质	不可以
MEDIA_UNMOUNTED	有介质，未挂载，在系统中删除	不可以
MEDIA_BAD_REMOVAL	介质在挂载前被移除，直接取出 SD 卡	不可以
MEDIA_CHECKING	正在磁盘检查，刚装上 SD 卡时	不可以
MEDIA_SHARED	SD 卡存在但没有挂载，并且通过 USB 大容量存储共享，操作打开 USB 存储	不可以
MEDIA_MOUNTED_READ_ONLY	SD 卡存在并且已挂载，但是挂载方式为只读	不可以
MEDIA_NOFS	介质存在但是为空白或用在不支持的文件系统	不可以
MEDIA_UNMOUNTABLE	存在 SD 卡但是不能挂载，例如发生在介质损坏时	不可以

通过表 9.4 可以看出，如果要读写 SD 卡，需要 Environment. getExternalStorageState(). equals(Environment. MEDIA_MOUNTED)值为 true 的结果。然后就可以获取 SD 卡根路径，后续操作与 J2SE 中对文件操作基本一致，可以调用 File 类的一系列方法新建、删除、查看文件属性等。

新建项目 part09_3，演示读写 SD 卡的操作，其中读写方法如代码 09-4 所示。布局文件与项目 part09_2 一样，只是本项目会将数据保存在 SD 卡中。对于 SD 卡的常规操作，如创建文件夹、创建文件、检验文件状态、复制文件等，应该维护成一个工具类，在第 10 章介绍网络编程时会给出一个比较简易的工具类，用于处理 SD 卡文件操作。

代码 09-4

```
//读取文件
private void read(String dirName,String fileName) {
    //判断 SD 卡状态是否可用
        if(!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){ ①
            Toast.makeText(this, "SD 卡状态异常", Toast.LENGTH_LONG).show();
            return;
    }
    //获取 SD 卡根路径
    File sdRoot = Environment.getExternalStorageDirectory();                    ②
    File file = new File(sdRoot.getAbsolutePath() + dirName,fileName);
    if(!file.exists()){
        Toast.makeText(this, "联系人数据不存在", Toast.LENGTH_LONG).show();      ③
        return;
    }
```

```
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(file));
            String ln = null;
            tv1.setText("");                                      ④
            while((ln = br.readLine())!= null){
                tv1.append(ln + "\n");
            }
        } catch (FileNotFoundException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        }finally{
            if(br!= null)
                try {
                    br.close();
                } catch (IOException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                }
        }
    }
    //保存文件
    private void save(String dirName,String fileName) {
        //判断 SD 卡状态是否可用
        if(!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
            Toast.makeText(this, "SD 卡状态异常", Toast.LENGTH_LONG).show();   ①
            return;
        }
        //获取 SD 卡根路径
        File sdRoot = Environment.getExternalStorageDirectory();   ②
        //创建文件夹
        File dir = new File(sdRoot,dirName);
        if(!dir.exists()){
            dir.mkdirs();                                          ③
        }
        //创建文件,并写入数据
        File file = new File(dir,fileName);
        PrintWriter pw = null;
        try {
            if(!file.exists()){
                file.createNewFile();
            }                                                      ④
            pw = new PrintWriter(new FileOutputStream(file,true));
            pw.println("好友姓名:" + et1.getText().toString() + "
            联系方式: " + et2.getText().toString());
            pw.flush();
```

```
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally{
            if(pw!= null)pw.close();
        }
    }
```

对于 Android 系统 SD 卡读写操作一般需要 4 个步骤(如代码 09-4 所标识的)。第一，判断 SD 卡状态；第二，获取根路径；第三，创建文件；第四，使用 I/O 的知识进行读写操作。

本项目运行效果如图 9.2 所示，但数据存储在 SD 卡中。打开 DDMS 视图，选择 File explorer 选项卡，依次展开 mnt→sdcard(SD 卡)→contract(代码中创建的文件夹)→data(代码中创建的文件夹)，文件 cd.txt 即为本项目所创建的文件，如图 9.4 所示。

mnt		2013-08-30	02:59	drwxrwxr-x
asec		2013-08-30	02:59	drwxr-xr-x
obb		2013-08-30	02:59	drwxr-xr-x
sdcard		2013-08-30	04:36	d---rwxr-x
Android		2013-08-29	04:36	d---rwxr-x
DCIM		2013-08-29	04:30	d---rwxr-x
LOST.DIR		2013-06-14	01:04	d---rwxr-x
contract		2013-08-30	04:36	d---rwxr-x
data		2013-08-30	04:36	d---rwxr-x
cd.txt	37	2013-08-30	04:36	----rwxr-x

图 9.4　SD 卡文件目录结构图

在 Android 中选择文件存储数据时，需要注意以下几点建议。

- 应用程序安装路径下存储文件不能过大，对于需要下载的文件(如歌曲、视频等)应该存放在 SD 卡上。
- 使用 SD 卡前需要检验 SD 卡的状态。
- 读写 SD 卡需要权限＜uses-permission android:name＝"android.permission.WRITE_EXTERNAL_STORAGE"/＞。

9.3　SQLite 数据库

SQLite 数据库不同于 Oracle、MySQL 等数据库，它是一款轻量级的关系型数据库，实际上就是一个文件，但支持 SQL 语句。由于 SQLite 占用的资源非常少，所以在很多嵌入式设备中都是用 SQLite 来存储数据的。Android 系统提供了丰富的 API 用于操作 SQLite 数据库。

9.3.1　SQLiteDatabase

Android 系统中用于操作数据库的类位于 android.database 和 android.database.sqlite 包中。SQLiteDatabase 类位于 android.database.sqlite 包中，它表示一个 SQLite 数据库，

可以进行创建、删除、执行 SQL 语句等操作。

SQLiteDatabase 提供了静态方法，用于得到 SQLiteDatabase 对象，常用的静态方法如下。

(1) openDatabase (String path, SQLiteDatabase. CursorFactory factory, int flags)，打开 path 指定的 SQLite 数据库。

(2) openOrCreateDatabase(String path, SQLiteDatabase. CursorFactory factory)，打开或创建 path 指定的 SQLite 数据库。

(3) openOrCreateDatabase(File file, SQLiteDatabase. CursorFactory factory)，打开或创建 File 所指定的 SQLite 数据库。

得到 SQLiteDatabase 数据库的对象后，就可以执行增、删、改、查各种操作了。实际上，SQLiteDatabase 类似于 JDBC 编程中的 Connection、Statement 和 PreparedStatement 的组合。SQLiteDatabase 常用的方法如表 9.5 所示。

表 9.5 SQLiteDatabase 常用的方法

操作分类	返回值类型	方　　法	说　　明
插入	long	insert(String table, String nullColumnHack, ContentValues values)	向 table 表中插入 values
删除	int	delete(String table, String whereClause, String[] whereArgs)	根据指定字段 whereClause 删除表 table 中的值，这些字段的取值是 whereArgs
修改	int	update (String table, ContentValues values, String whereClause, String[] whereArgs)	更新表 table，更新内容是 values，更新条件由 whereClause 和 whereArsg 决定
查询	Cursor	query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)	查询表 table 中的字段 columns，条件由 selection 和 selectionArgs 组合，并由其他参数决定成组、排序、分页等 SQL 操作
	Cursor	query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	同上
执行 SQL	void	execSQL(String sql)	执行 SQL 语句
	void	execSQL(String sql, Object[] bindArgs)	执行带有占位符? 的 SQL 语句
其他操作	void	beginTransaction()	开启事务控制
	void	endTransaction()	结束事物
	void	close()	关闭数据库

SQLiteDatabase 可以直接执行 SQL 语句，也可以将增、删、改、查分配到各方法执行。仔细观察表 9.5 中的各种操作方法，虽然参数众多，但组合起来无非 SQL 语句而已，Android 之所以这样做是担心某些程序员对 SQL 语句不熟悉。

上述方法中涉及两个类 ContentValues 和 Cursor 的使用。ContentValues 位于 android. content 包中，它实际上就如同一个 Map 集合，提供丰富的 put 和 get 操作，用于存放键值对。其中，键就是表中的某一个字段，值就是该字段对应的内容。Cursor 是一个接

口，位于 android.content 包，它的功能类似于 JDBC 中的 ResultSet，主要用于查询结果集。Cursor 的常用操作如表 9.6 所示。

表 9.6 Cursor 的常用操作

返回值类型	方 法 名 称	方 法 描 述
int	getCount()	总记录条数
int	getColumnCount()	返回列数
boolean	isFirst()	判断是否第一条记录
boolean	isLast()	判断是否最后一条记录
boolean	moveToFirst()	移动到第一条记录
boolean	moveToLast()	移动到最后一条记录
boolean	move(int offset)	移动到指定的记录
boolean	moveToNext()	移动到下一条记录
boolean	moveToPrevious()	移动到上一条记录
int	getColumnIndex(String columnName)	获得指定列的索引值，从 0 开始，如果返回 -1 说明不存在此列
String	getColumnName(int columnIndex)	返回指定索引的列名
String[]	getColumnNames()	返回 Cursor 中所有列的集合
×××	get×××(int columnIndex)	一系列这种方法，用于获取指定下表列的值

9.3.2 数据库的基本操作

SQLiteDatabase 提供两种方式完成数据库的基本操作，一种是直接执行 SQL，另一种是执行对应的方法。新建项目 part09_4，演示 SQLiteDatabase 的基本用法。在该项目中，activity_main.xml 为主界面布局文件，如图 9.5 所示，listview_item.xml 是 ListView 控件所用布局文件，完整代码请查阅随书配套资料。

创建数据库、新建表的方法如代码 09-5 所示。

代码 09-5

```
//创建数据库
public SQLiteDatabase getDb(String dbPath,String dbName){
    File dbDir = new File(Environment.getExternalStorageDirectory(),dbPath);
    if(!dbDir.exists()){
        dbDir.mkdirs();
    }
    File dbFile  = new File(dbDir,dbName);
    try {
        if(!dbFile.exists()){
            dbFile.createNewFile();
        }
        sdb = SQLiteDatabase.openOrCreateDatabase(dbFile, null);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return sdb;
```

```
    }
    //创建数据库表
    public void createTable(String tbName){
        //SQLite 基本上与 MySQL 类似
        String sql = "create table " + tbName + "(_id integer primary key autoincrement," +
        "bookname varchar,bookprice float)";
        try {
            //同名的 table 只能存在一个,如果重复一定会出现异常
            sdb.execSQL(sql);
        } catch (Exception e) {
            Log.i("Tag", "该表已存在!");
            //e.printStackTrace();
        }
    }
```

方法 openOrCreateDatabase 在执行时，如果所指定的数据库文件不存在则主动创建，否则直接打开该数据库文件。SQLite 数据库的后缀名一般设为 db3 或 db。

SQLiteDatabase 可以直接执行 SQL 语句，如果需要在 SQLite 数据库中创建表时，可以执行创建语句。此处需要注意的是，如果重复执行创建相同表名的语句，会有异常发生。所以代码 09-5 中加粗语句需要放在 try 块中执行。这种方式虽然能够解决程序崩溃的问题，但很显然，做法不是很优雅。在后面会介绍使用 SQLiteOpenHelper 创建数据库，将会解决这个问题。

上述代码中还有一点需要特别注意，在创建表字段时，主键为_id，这是为了方便后面使用 SimpleCursorAdapter 向 ListView 列表填充数据。SimpleCursorAdapter 填充数据时，要求其主键必须为_id，否则抛出异常。

代码 09-6 是插入数据的方法。使用 insert 向 SQLite 数据库中插入数据时，需要先构建 ContentValues，其中的键是相应表中的字段值，二者要一致。

代码 09-6

```
    //插入数据
    public void insertData(){
        String bName = et1.getText().toString();
        String bPrice  = et2.getText().toString();
        //创建 ContentValues,存放数据
        ContentValues cv = new ContentValues();
        cv.put("bookname", bName);
        cv.put("bookprice", bPrice);
        long r  = sdb.insert("tb_books", null,cv);
        Log.i("Tag", "插入数据成功 ID = " + r);
    }
```

更新相应字段的值时，也需要先构建 ContentValues，如代码 09-7 所示。更新方法 update("tb_books", cv, "_id=?", new String[]{id})，意在向表 tb_books 中进行更新，更新的字段和值由 ContentValues 确定，更新的条件是"_id=?"，此处的写法与 SQL 语句中的 where 子句一致，占位符的值由后面的字符数组确定，之所以是数组，就是考虑了可能有多

图 9.5　SQLite 基本操作运行效果图

个占位符的情况。

📖 **代码 09-7**

```
//根据 ID 更新数据
public void updateData(){
    String id  = et3.getText().toString();
    String bPrice  = et4.getText().toString();
    if(id.length()<1)return;
    ContentValues cv = new ContentValues();
    cv.put("bookprice", bPrice);
    long r = sdb.update("tb_books", cv, "_id = ?", new String[]{id});
    Log.i("Tag", "更新数据成功 ID = " + r);
}
```

删除数据的方式相对简单，如代码 09-8 所示。方法 delete("tb_books"，"_id=?"，new String[]{id})指明删除表 tb_books 中的数据，条件是由_id 决定，值由后面的数组决定。

📖 **代码 09-8**

```
//根据 ID 删除数据
public void deleteData(){
    String id = et5.getText().toString();
    if(id.length()<1)return;
    int r = sdb.delete("tb_books", "_id = ?", new String[]{id});
    Log.i("Tag", "删除数据成功 ID = " + r);
}
```

查询数据会返回 Cursor 对象，利用它可以直接创建 SimpleCursorAdapter 适配器，如代码 09-9 所示。query 方法中的参数相对较多，读者可以对照标准的 SQL 查询语句来理解，将 where、group by、order by 等子句对应成相应的参数即可，尤其注意参数和值的匹配要对称。

代码 09-9

```
//根据 ID 查询数据,如果 ID 为 null,则查询全部数据
public Cursor queryData(){
    String id = et6.getText().toString();
    Cursor c = null;
    if(id.length()< 1){
        c = sdb.query("tb_books", new String[]{"_id","bookname","bookprice"},
                null,null,null, null,null);
    }else{
        c = sdb.query("tb_books", new String[]{"_id","bookname","bookprice"},
                "_id = ?",new String[]{id},null, null,null);
    }
    return c;
}
private void showData() {
    Cursor c = queryData();
    CursorAdapter ca = new SimpleCursorAdapter(this,R.layout.listview_item,c,new String[]{"
_id",
            "bookname","bookprice"},new int[]{R.id.listview_item_id,R.id.listview_item_
name,
            R.id.listview_item_price},CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
    lv.setAdapter(ca);
}
```

至于 SimpleCursorAdapter 适配器中各参数的含义,可以查阅前面章节中对数据适配器的介绍,参数含义基本上是一致的。

打开的 DDMS 视图如图 9.6 所示,找到项目中创建的数据库,通过右上角的 pull 按钮将其复制到 PC 中,可以借助 SQLite 工具查看该数据库中的数据。在 android-sdk\tools 目录下有一个可执行文件 sqlite3.exe,该工具即为 Android 提供用于管理 SQLite 数据库的。它类似于 MySQL 的命令行界面,可以执行常规的操作。

sdcard		2013-08-30	10:13	d---rwxr-x
Alarms		2013-08-30	09:52	d---rwxr-x
DCIM		2013-08-30	09:52	d---rwxr-x
Download		2013-08-30	09:52	d---rwxr-x
LOST.DIR		2013-08-30	09:51	d---rwxr-x
Movies		2013-08-30	09:52	d---rwxr-x
Music		2013-08-30	09:52	d---rwxr-x
Notifications		2013-08-30	09:52	d---rwxr-x
Pictures		2013-08-30	09:52	d---rwxr-x
Podcasts		2013-08-30	09:52	d---rwxr-x
Ringtones		2013-08-30	09:52	d---rwxr-x
database		2013-08-30	10:19	d---rwxr-x
booksdb.db3	5120	2013-08-30	10:33	----rwxr-x
booksdb.db3-journal	0	2013-08-30	10:33	----rwxr-x

图 9.6 在 SD 卡创建的数据库

9.3.3 SQLite 管理工具

SQLiteSpy 也是用于管理 SQLite 数据库的工具，只有 2MB 大小，不同于 sqlite3.exe，它是图形化操作界面，如图 9.7 所示。通过 File 菜单下的 Open database 就可以打开 SQLite 数据库，当然也可以创建 SQLite 数据库。该款软件的使用比较简单，在此不做过多说明。

需要注意的是，SQLite 数据库中的字段类型为 INTEGER、REAL（浮点型）、TEXT（文本）和 BLOB（二进制），但也可以接受 varchar(n) 和 char(n) 类型，只不过在存储数据时也会自动转型为上述类型。除了声明为 INTEGER Primary key 的主键只能为整型之外，SQLite 对类型的限制并不严格。

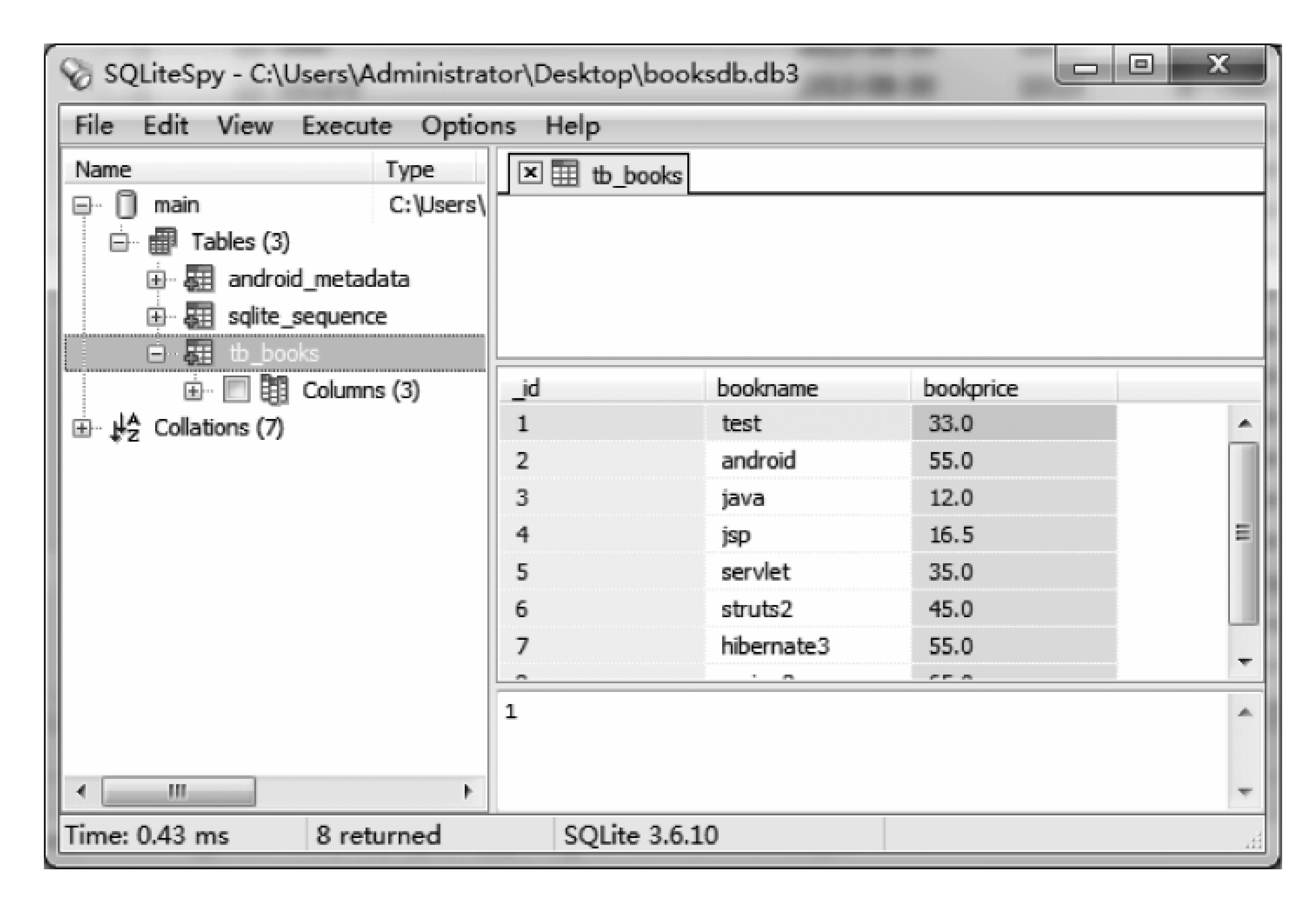


图 9.7 SQLiteSpy 操作主界面

除了 SQLiteSpy 用于管理 SQLite 数据库之外，还可以选择 SQLiteExpert 管理软件，它要比 SQLiteSpy 功能更加丰富，该软件可以从 http://www.sqliteexpert.com/download.html 免费下载，运行界面如图 9.8 所示。借助它可以很容易查看已有 SQLite 数据库，创建新数据库、新建表等。

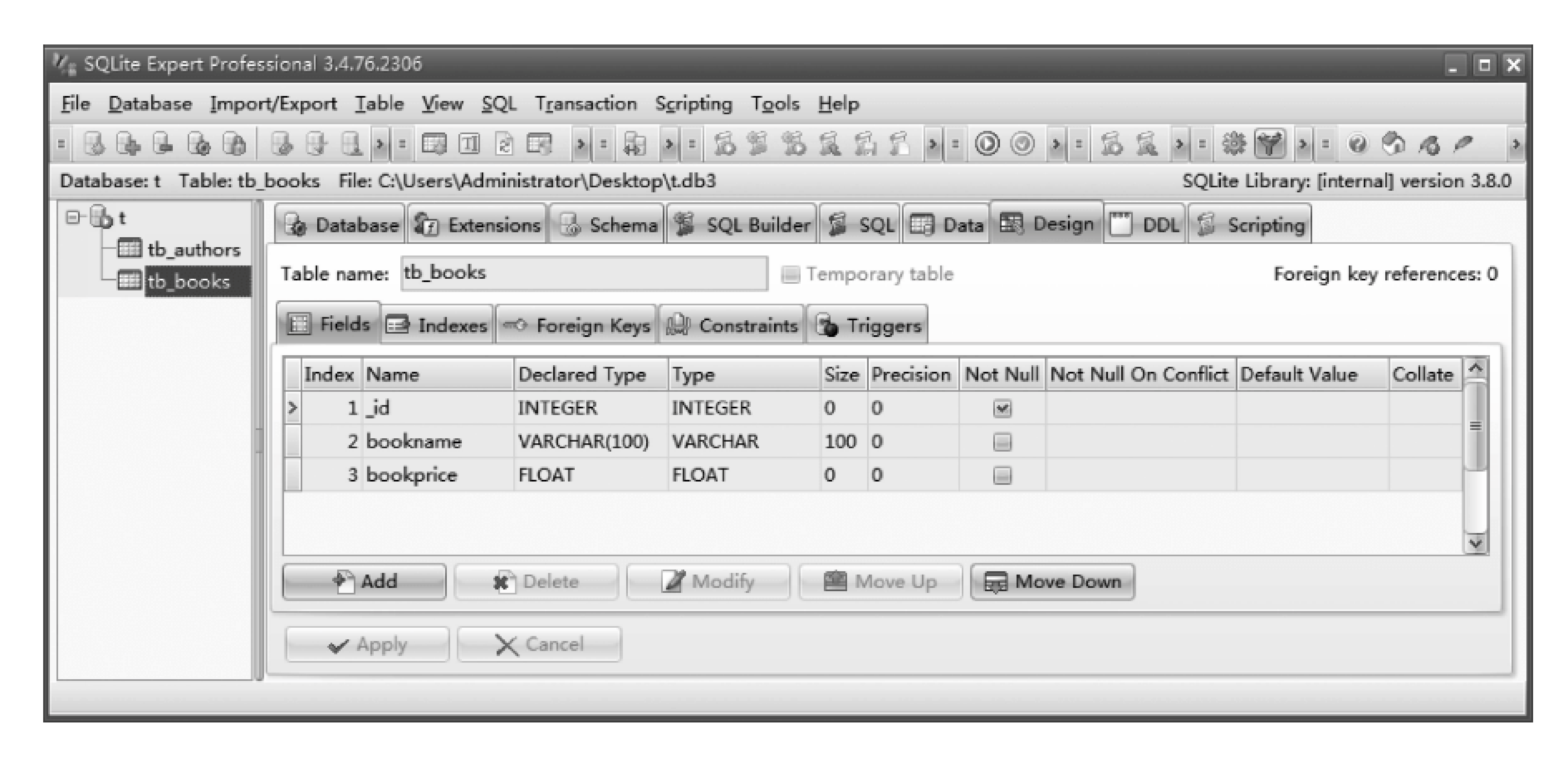


图 9.8 SQLiteExpert 主界面

9.3.4 SQLiteOpenHelper

在前面的项目中,执行新建数据库表时,如果该表已经存在,则会抛出异常,虽然已经将其放入 try 块语句中,但这种做法显然不是很好。SQLiteOpenHelper 位于 android.database.sqlite 包中,是一个抽象类,它主要作为 SQLiteDatabase 的一个帮助类,用来管理数据库的创建和版本的更新。常用的做法是建立一个类继承它,并实现它的 onCreate 和 onUpgrade 方法。SQLiteOpenHelper 类的常用方法详见表 9.7。

表 9.7 SQLiteOpenHelper 类的常用方法

方　法　名	方 法 描 述
SQLiteOpenHelper(Context context,String name,SQLiteDatabase.CursorFactory factory,int version)	构造方法,一般是传递一个要创建的数据库名称参数
onCreate(SQLiteDatabase db)	创建数据库时调用,是抽象方法,一般将创建表等初始化操作在该方法中执行
onUpgrade(SQLiteDatabase db,int oldVersion , int newVersion)	版本更新时调用,是抽象方法
getReadableDatabase()	创建或打开一个只读数据库
getWritableDatabase()	创建或打开一个读写数据库

需要重点说明的是 onCreate 方法和 onUpgrade 方法。当通过 get 方法获取 SQLiteDatabase 数据库时,如果所指定的数据库不存在,则需要创建该数据库,执行 onCreate 方法,当所指定数据库存在时,直接返回该数据库。也就是说,onCreate 方法只有在数据库不存在,需要新建数据库时才会执行,因此该方法中主要进行数据库表的创建。

onUpgrade 方法用于更新数据库,更新数据库的标志是数据库的版本 version,该参数由程序自行控制,但创建 SQLiteOpenHelper 时传入的参数 version 大于以前版本,onUpgrade 方法便会执行,因此可以在该方法中处理软件升级时对数据库结构的更改操作。

新建 MySQLiteOpenHelper 继承 SQLiteOpenHelper,如代码 09-10 所示。在创建 SQLiteDatabase 时可以通过语句 new MySQLiteOpenHelper(this,"ndb.db3",null,1).getWritableDatabase()实现。此处数据库名 ndb.db3 不能包含路径,可以采用 Environment.getExternalStorageDirectory().getAbsolutePath()+File.separator+"ndb.db3"方式来实现在 SD 卡上创建数据库。

📖代码 09-10

```
public class MySQLiteOpenHelper extends SQLiteOpenHelper {
    public MySQLiteOpenHelper(Context context, String name,
            CursorFactory factory, int version,
            DatabaseErrorHandler errorHandler) {
        super(context, name, factory, version, errorHandler);
    }
    public MySQLiteOpenHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
```

```
        }
        @Override
        public void onCreate(SQLiteDatabase sdb) {
            Log.i("Tag", "onCrate 执行,创建表");
            //新建表 tb_books
            String sql = "create table tb_books(" +
            "_id integer primary key autoincrement," +
            "bookname varchar,bookprice float,authorID integer)";
            sdb.execSQL(sql);
            //新建表 tb_authors
            sql = "create table tb_authors(" +
            "_id integer primary key autoincrement," +
            "authorname varchar,authorphone varchar)";
            sdb.execSQL(sql);
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            Log.i("Tag", "SQLiteDatabase upgrade " + oldVersion + " > " + newVersion);
        }
    }
```

习　　题

1. 方法 getSharedPreferences (String name, int mode)中参数 mode 有何作用？可以取哪些值？
2. 使用 SharedPreferences 保存的数据具有什么样的特点？保存文件的路径在哪里？
3. 设计开发手机下载软件，最多允许 5 个文件同时下载。
4. SQLiteDatabase 常用的操作有哪些？如何使用？
5. SQLiteOpenHelper 类的作用是什么？



CHAPTER 10

第10章

网络编程

本章主要内容：

TCP 通信与 Socket 应用；
URL 访问网络资源；
HTTP 请求；
HttpClient 应用；
WebService 应用；
XML 解析；
JSON 解析。

本章主要介绍 Android 系统网络编程相关知识，涉及 Android 与 PC 或服务器通信、发送请求、获取数据的内容。Android 系统直接引入了 JDK 网络编程的知识，所以在 Android 中完全可以直接进行基于 TCP 或 UDP 的 Socket 编程或 DatagramSocket 编程。Android 系统可以使用 URL 访问网络资源，发起 Http 请求操作，通过内置的 HttpClient 访问受保护的网络资源。

本章还涉及如何使用 WebService，解析第三方提供的数据，解析 XML，解析 JSON 等内容。

10.1 基于 TCP 的通信

TCP(Transmission Control Protocol，传输控制协议)是面向连接的、可靠的、基于字节流的传输层(Transport layer)通信协议。

10.1.1 TCP 与 Socket 编程

所谓 Socket 编程就是基于 TCP 的网络编程，也可以说，Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。它是 TCP/IP 网络所提供的 API，可以用来开发 TCP/IP 网络上的应用程序。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对开发者来说，一组简单的接口就是全部，而无须

担心数据的格式。Socket 会自动组织数据，以符合指定的协议。

10.1.2 ServerSocket 与 Socket

Socket 编程是比较简单的，在网络通信开始前，需要建立一个连接，这就需要用到 Socket 和 ServerSocket 类，它们位于 java.net 包中。ServerSocket 用于服务器端，Socket 是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个 Socket 实例，操作这个实例，完成所需的会话，如图 10.1 所示。

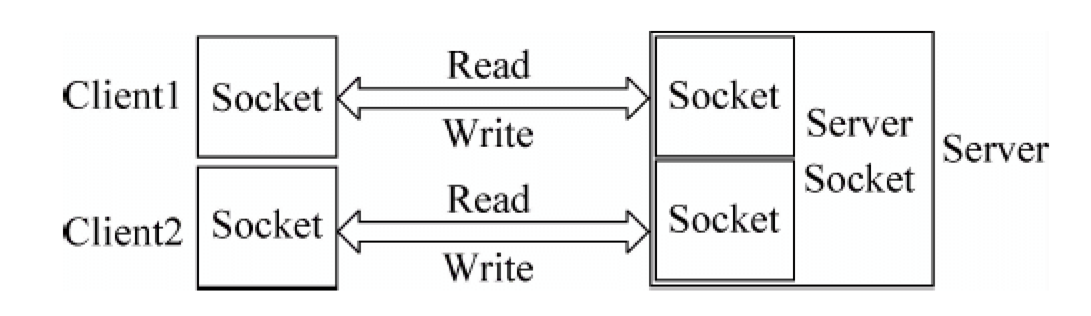


图 10.1 Socket 通信示意图

服务器端使用 ServerSocket 监听指定的端口（端口号通常指定为 1024 以后的），等待客户连接请求，一旦客户端连接后，会话便产生；在完成会话后，需要关闭连接。客户端使用 Socket 对网络上某一个服务器的某一个端口发出连接请求（该服务器的该端口有监听服务），一旦连接成功，便开始会话；会话完成后，需要关闭 Socket。Socket 编程时常用方法如下。

（1）ServerSocket 类的监听方法是 accept()，该方法用于产生"阻塞"，直到接收到一个连接，并且返回一个客户端的 Socket 对象实例。"阻塞"是一个术语，它使程序运行暂时"停留"在这个地方，直到一个会话产生，然后程序继续；通常"阻塞"是由循环产生的。

（2）getInputStream 方法获得网络连接输入，同时返回一个 IutputStream 对象实例。

（3）getOutputStream 方法连接的另一端将得到输入，同时返回一个 OutputStream 对象实例。

Android 操作系统完全引入了 java.net 包，所以可以像 J2SE 中那样进行 Socket 编程。下面演示如何在 Android 平台进行 Socket 通信。新建 Android 项目 part10_1，作为手机客户端，新建 Java 项目 part10_1_1，作为服务器端。需要注意的是，Android 中无法直接在子线程中修改 UI，而 Socket 通信时肯定需要子线程监听服务端转发的消息，所以只能借助 Handler 间接修改 UI。该项目演示手机客户端登录服务器，实现多人聊天的功能，基本功能可以参考项目代码。

该项目布局比较简单，包含一个 EditText 用于编辑文字，一个 Button 用于发送数据，一个 TextView 用于显示聊天记录，TextView 需要放在 ScrollView 中使用，以实现滚动功能。Activity 代码如代码 10-1 所示，服务器端代码请查阅随书配套资料。

📖 代码 10-1

```
public class MainActivity extends Activity {
    Button send;
    EditText txt;
    TextView content;
    Socket socket;          //引用 Socket
```

```
boolean flag = true;
//创建 Handler
Handler handler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        Bundle b = msg.getData();
        content.append(b.getString("txt") + "\n");
    }
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    send = (Button) findViewById(R.id.send);
    txt = (EditText) findViewById(R.id.txt);
    content = (TextView) findViewById(R.id.content);
    send.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View v) {
            sendMsg();
        }
    });
    //初始化 Socket
    initSocket();
    Log.i("Tag", "初始化 socket 结束, socket 状态" + socket.isConnected());
}
//创建线程处理,接受服务器信息
private void initReceiver() {
    new Thread(new Runnable(){
        @Override
        public void run() {
            if(socket == null)return;
            try {
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
                while(flag){
                    final String str = br.readLine();
                    if(str!= null&&str.length()> 0){
                        Log.i("Tag", "收到服务器 消息:" + str);
                        handler.post(new Runnable(){
                            @Override
                            public void run() {
                                Message m = new Message();
                                Bundle b = new Bundle();
                                b.putString("txt", str);
                                m.setData(b);
                                handler.sendMessage(m);
                                Log.i("Tag", "handler 抛出消息");
                            }
```

```
                        });
                    }
                    Thread.sleep(100);
                }
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }).start();
    Log.i("Tag", "接收线程已启动");
}
//初始化 Socket 连接
private void initSocket() {
    try {
        socket = new Socket("192.168.0.10",6677);
        Log.i("Tag", "Socket 创建成功!");
    } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
//发送消息
public void sendMsg(){
    String txts = txt.getText().toString();
    if(txts.length()< 1)return;
    if(socket == null)return;
    PrintWriter pw = null ;
    try {
        pw = new PrintWriter(socket.getOutputStream());
        pw.println(txts);
        pw.flush();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
@Override
protected void onStart() {
    super.onStart();
    if(socket == null){
```

```
            Toast.makeText(this, "网络连接错误,Socket 异常!请开启网络连接,重新运行本软
件",Toast.LENGTH_LONG).show();
            return;
        }
        //初始化接听
        initReceiver();
        Log.i("Tag", "初始化 接收监听器接收");
    }
    @Override
    protected void onPause() {
        super.onPause();
        flag = false;
    }
}
```

在测试时请先运行服务器代码,再执行移动端程序。代码 10-1 涉及 Socket 通信的主要是方法 sendMsg()和方法 initReceiver(),下面对这两个方法详细介绍。

sendMsg 用于发送数据,当单击“发送”按钮时会执行该方法。通过 Socket 发送数据比较简单,只需要得到输出流即可,考虑到只发送文本信息,代码中将输出流包装为 PrintWriter 使用。(Socket 常用方法属于 J2SE 的基础知识,此处不再介绍,如果对该部分内容比较陌生可以查阅相关资料。)

initReceiver 用于接收服务器端发送过来的数据,该方法需要开启子线程。考虑到 Activity 的生命周期,该方法建议在 onStart 中调用。子线程一直监听服务器端是否有数据到达,如果接收到数据,封装到 Message 对象中,使用 Handler 抛到主 UI 线程中,随即呈现在 TextView 中。

注意,在运行上述项目时需要添加网络权限:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

10.2 URL 获取网络资源

统一资源定位符(Uniform Resource Locator,URL)相当于一个文件名在网络范围的扩展,是与网络相连的机器上的任何可访问对象的一个指针。通常,URL 的一般形式是:<URL 的访问协议>://<主机>:<端口>/<路径>,目前支持的协议有 File(读取文件)、FTP(文件传输协议)、HTTP(超文本传输协议)、HTTPS(安全 HTTP)、JAR(读取 JAR 文件)。

10.2.1 URL 介绍

URL 位于 java.net 包中,是 JDK 所提供的类,在 Android 平台可以直接使用。URL 提供了较多构造方法,根据不同参数创建 URL 对象。URL 类常用方法如表 10.1 所示。

表 10.1 URL 类常用方法

返回值类型	方　　法	说　　明
String	getFile()	返回 URL 中资源名称
String	getHost()	返回 URL 的主机名或 IP 地址
String	getPath()	返回 URL 中路径部分字符串
int	getPort()	返回 URL 中的端口号
String	getProtocol()	返回 URL 所采用的访问协议
String	getQuery()	返回 URL 的请求字符串，即？后面的字符串
URLConnection	openConnection()	返回到 URL 指定资源的连接对象，类似于数据库连接的 Connection
InputStream	openStream()	返回到 URL 指定资源的输入流，此输入流也可以通过 URLConnection 的 getInputStream 方法获取

使用 URL 访问网络中的资源非常简单，与 I/O 中介绍的文件复制类似。新建项目 part10_2，演示使用 URL 获取网络中的资源，并将其下载到 SD 卡。该项目需要开启以下两个权限。

(1) <uses-permission android:name="android.permission.INTERNET"/>，访问网络。

(2) <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>，读写 SD 卡。

代码 10-2

```
public class MainActivity extends Activity {
    ImageView iv;           //预览图片
    Bitmap bitmap;          //图片
    URL url;                //URL 地址
    //更新 UI
    Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            if(msg.what == 0x100){
                iv.setImageBitmap(bitmap);
            }else if(msg.what == 0x200){
                Toast.makeText(MainActivity.this,
                        "图片下载完成", Toast.LENGTH_LONG).show();
            }
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        iv = (ImageView) findViewById(R.id.imageView1);
        //本代码只为测试 URL 下载资源
        //开启子线程
```

```
            new Thread(new Runnable(){
                @Override
                public void run() {
                    try {
                        //资源的地址可以随机修改
                        url = new URL("http://image1.nbd.com.cn/uploads/articles/thumbnails
                                /28637/123.x_large.jpg");
                        //获取输入流
                        InputStream is  = url.openStream();
                        //解析成图片
                        bitmap  = BitmapFactory.decodeStream(is);
                        Log.i("Tag", "解析图片结束");
                        //发送消息,通知 UI 更改
                        handler.sendEmptyMessage(0x100);
                        is.close();
                        //将该图片下载到 SD 卡
                        is = url.openStream();
                        File file  = new SaveDataSDCard().writeData("download", "andr01.jpg", is);
                        Log.i("Tag", "下载图片结束");
                        if(file!= null){
                            handler.sendEmptyMessage(0x200);
                        }
                        is.close();
                    } catch (MalformedURLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }).start();
        }
    }
```

由于联网下载资源比较耗时,建议将其放在子线程中进行。Android 操作系统的新版本几乎都会强制联网程序写在子线程中,否则会抛出异常。对于在开发过程中经常使用的类,可以写成工具类,方便调用,如代码 10-2 中对 SD 卡中数据存储操作的类 SaveDataSDCard,详见代码 10-3,读者可以自行修改,以便完善此工具类。

代码 10-3

```
/*
 * 用于向 SD 卡保存数据
 *  */
public class SaveDataSDCard {
    final static String T = "Tag";
    //获取 SD 卡根路径
    public File getRoot(){
```

```
        if(Environment.getExternalStorageState()
                .equalsIgnoreCase(Environment.MEDIA_REMOVED)){
            Log.i(T, "Sd 不存在");
            return null;
        }
        return Environment.getExternalStorageDirectory();
    }

    //创建文件夹
    public File createDir(String dirStr){
        File dir = new File(getRoot(),dirStr);
        if(!dir.exists()){
            dir.mkdirs();
        }
        return dir;
    }
    //创建文件
    public File createFile(String filePath,String fileName){
        File dir = createDir(filePath);
        File file = new File(dir,fileName);
        try {
            if(!file.exists()){
                file.createNewFile();
            }
        } catch (IOException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        }
        return file;
    }
    /* 向指定文件中写入数据,数据为一个字符串
     * path:文件路径
     * fileName: 文件名
     * content: 待写入字符串
     * */
    public File writeData(String path,String fileName,String content){
        File file = createFile(path,fileName);
        if(file == null)return null;
        PrintWriter pw = null;
        boolean r = false;
        try {
            pw = new PrintWriter(file);
            pw.println(content);
            pw.flush();
            r = true;
        } catch (FileNotFoundException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        }finally{
            pw.close();
```

```
        }
        return file;
    }
    /* 向指定文件中写入数据,数据是一个输入流
     * path:文件路径
     * fileName: 文件名
     * is: 输入流
     * */
    public File writeData(String path,String fileName,InputStream is){
        File file = createFile(path,fileName);
        if(file == null)return null;
        FileOutputStream fos = null;
        byte buffer[] = new byte[1024 * 4];
        int count;
        try {
            fos = new FileOutputStream(file);
            while( (count = is.read(buffer))> 0){
                fos.write(buffer, 0, count);
            }
            fos.flush();
        } catch (FileNotFoundException e) {

            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally{
            if(fos!= null)
                try {
                    fos.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
        return file;
    }
}
```

10.2.2 URLConnection 与 HttpURLConnection

使用 URL 可以获取相应的网络资源,但无法获取更多的控制信息,此时需要借助 URLConnection,该类位于 java.net 包中,它相当于 URL 资源与应用程序之间的连接桥梁,作用与 JDBC 中的 Connection 较为类似,可以借助它向 URL 发送请求数据,读取返回的资源。HttpURLConnection 是 URLConnection 的子类,主要是建立于 HTTP 上的连接,如通常的访问网页资源,可以直接使用 HttpURLConnection 类。

使用 URLConnection 建立网络资源连接时，可以参考以下步骤。

（1）调用 URL 类中 OpenConnection 方法，获取 URLConnection 对象，如果 URL 创建时采用 HTTP，则可以直接强制转换为 HttpURLConnection 对象。

（2）设置连接属性。

（3）调用 connect 方法，正式建立连接。

（4）建立连接后，使用方法 getHeaderFields() 和 getHeaderFieldKey(int posn) 可以查询头信息。

（5）获取输入流，读取返回资源，该输入流与在 URL 中获取的输入流一致。

URLConnection 中用于设置连接属性及获取输入输出流的方法参考表 10.2。

表 10.2　URLConnection 常用的方法

方　　法	说　　明
setDoInput(boolean newValue)	是否允许输入，如果需要向 URL 发送数据，参数应设置为 true
setDoOutput(boolean newValue)	设置允许输出，与上面的方法都用于设置头字段
setUseCaches(boolean newValue)	设置是否缓存，涉及 useCaches 字段的值
setAllowUserInteraction (boolean newValue)	设置 allowUserInteraction 字段的值
setRequestProperty(String field, String newValue)	设置相应字段的值，如 setRequestProperty(accept, */*)
setConnectTimeout(int timeout)	设置连接超时的上限
setReadTimeout(int timeout)	设置读取数据时间上限，如果时间超过上限，会抛出异常
getOutputStream()	获取输出流，用于向 URL 指定资源发生请求参数
getInputStream()	获取输入流，读取 URL 返回资源

HttpURLConnection 常用的方法参考表 10.3。

表 10.3　HttpURLConnection 常用的方法

方　　法	说　　明
getResponseCode()	获取服务器响应代码，如 200 表示 HTTP_OK，404 表示 HTTP_NOT_FOUND
getResponseMessage()	获取服务器返回的消息
getRequestMethod()	获取请求方法，GET 或 POST
setRequestMethod(String method)	设置请求方式，GET 或 POST

在使用 URLConnection 或 HttpURLConnection 访问指定资源时，应该注意以下几点建议。

- 如果需要向 URL 发出请求参数，应该先使用输出流，后使用输入流。
- 联网的程序最后写在子线程中，避免主线程被阻塞。
- Android 操作系统的不同版本对联网线程的规定不尽一致，在使用时需要注意版本问题。

10.2.3 Get 请求与 Post 请求

在进行 Web 应用程序开发时，经常需要用到的就是页面与服务器传递数据的方法 Get 或 Post。虽然二者都可以完成数据传递，但它们具有很大的区别。

第一，Get 将表单中的数据按照 variable=value 的形式，添加到 action 所指向的 URL 后面，并且两者使用？连接，而各个变量之间使用 & 连接；Post 是将表单中的数据放在 form 的数据体中，按照变量和值相对应的方式，传递到 action 所指向的 URL。

第二，Get 是不安全的，因为在传输过程中，数据被放在请求的 URL 中，是可见的（当然也可以采用加密方法），而如今现有的很多服务器、代理服务器或者用户代理都会将请求 URL 记录到日志文件中，然后放在某个地方，这样就可能会有一些隐私的信息被第三方看到。另外，用户也可以在浏览器上直接看到提交的数据，一些系统内部消息将会一同显示在用户面前。Post 的所有操作对用户来说都是不可见的。

第三，Get 传输的数据量小，这主要是因为受 URL 长度限制，不同浏览器大小限制有所不同，一般不超过 2KB 都不会出问题；而 Post 可以传输大量的数据，所以在上传文件时只能使用 Post。

第四，Get 限制 Form 表单的数据集的值必须为 ASCII 字符；而 Post 支持整个 ISO 10646 字符集。默认是用 ISO-8859-1 编码。

下面新建项目 part10_3，演示在移动端发起 Get 请求和 Post 请求，并传递参数的方式。本次测试需要涉及 Web 服务器，由项目 part10_3_3 实现（详见随书配套资料），该项目代码由 MyEclipse 9 开发，在运行时需要部署到服务器，如 Tomcat。（关于 Web 应用程序开发的相关内容，请学习其他资料。）

手机端程序的布局比较简单，只有两个按钮，分别发送 Get 请求和 Post 请求，完整的布局文件代码请查阅随书配套资料。MainActivity.java 源代码请查看代码 10-4。

代码 10-4

```
public class MainActivity extends Activity implements OnClickListener {
    Button b1,b2;
    TextView tv;
    String urlStr = "http://192.168.0.10:8080/part10_3_3/servlet/LoginServlet";
    Handler handler  = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            Bundle b  = msg.getData();
            tv.append(b.getString("msg") + "\n");
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1 = (Button) findViewById(R.id.bt_get);
```

```
        b2 = (Button) findViewById(R.id.bt_post);
        tv = (TextView) findViewById(R.id.tv);
        b1.setOnClickListener(this);
        b2.setOnClickListener(this);
    }
    @Override
    public void onClick(View arg0) {
        if(arg0.getId() == R.id.bt_get){
            //发起 Get 请求
            getRequest();
        }else {
            //发起 Post 请求
            postRequest();
        }
    }
    //Get 方式
    public void getRequest(){
        final String us = urlStr + "?loginName = admin&loginPwd = admin";
        new Thread (new Runnable(){
            @Override
            public void run() {
                BufferedReader br = null;
                try {
                    //Step1 创建 URL
                    URL url  = new URL(us);
                    //Step2 获取 HttpURLConnection 对象
                    HttpURLConnection httpConn = (HttpURLConnection) url.openConnection();
                    //Step3 设置属性
                    httpConn.setRequestProperty("accept", " * / * ");
                    httpConn.setDoInput(true);
                    httpConn.setDoOutput(true);
                    httpConn.setConnectTimeout(5000);
                    //Step4 连接
                    httpConn.connect();
                    //Step5 获取连接属性
                    int stat = httpConn.getResponseCode();
                    String ss = httpConn.getResponseMessage();
                    Log.i("Tag", "服务器返回代码 " + stat + " ," + ss);
                    //Step6 读取返回数据
                    String msg = "";
                    if(stat == 200){
                        br = new BufferedReader(
                                new InputStreamReader(httpConn.getInputStream()));
                        msg = br.readLine();
                    }else{
                        msg = "请求失败……";
                    }
                    Bundle b = new Bundle();
                    b.putString("msg", msg);
                    Message m  = new Message();
```

```
                        m.setData(b);
                        handler.sendMessage(m);
                    } catch (MalformedURLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }finally{
                        if(br!=null)
                            try {
                                br.close();
                            } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                    }
                }
            }).start();
        }
        //Post方式
        public void postRequest(){
            new Thread (new Runnable(){
                @Override
                public void run() {
                    PrintWriter pw = null;
                    BufferedReader br  = null;
                    try {
                        URL url  = new URL(urlStr);
                        HttpURLConnection httpConn = (HttpURLConnection) url.openConnection();
                        httpConn.setRequestProperty("accept", "*/*");
                        httpConn.setDoInput(true);
                        httpConn.setDoOutput(true);
                        httpConn.setConnectTimeout(5000);
                        //httpConn.connect();
                        //发送请求参数
                        pw = new PrintWriter(httpConn.getOutputStream());
                        pw.println("loginName=admin&loginPwd=admin");
                        pw.flush();
                        int stat = httpConn.getResponseCode();
                        String ss = httpConn.getResponseMessage();
                        Log.i("Tag", "服务器返回代码" + stat + "," + ss);
                        //读取响应
                        String msg = "";
                        if(stat == 200){
                                br = new BufferedReader(new InputStreamReader(httpConn.
getInputStream()));
                            msg = br.readLine();
                        }else{
                            msg = "请求失败……";
```

```
                    }
                    Bundle b = new Bundle();
                    b.putString("msg", msg);
                    Message m  = new Message();
                    m.setData(b);
                    handler.sendMessage(m);
                } catch (MalformedURLException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto - generated catch block
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

对比 Get 方式和 Post 方式发送请求，步骤基本基本一致，可以参考代码 10-4 中 1～6 的步骤，只是在请求参数的处理上不同。Get 方式需要将所有的请求参数附加到 URL 串后，Post 方式采用输出流直接输出。对于 Post 请求方式，httpConn.setDoInput(true)和 httpConn.setDoOutput(true)必须设定。另外，在处理输出流和输入流时，需要先使用输出流，然后使用输入流。项目运行效果如图 10.2 所示。



图 10.2　Get 请求与 Post 请求

手机端项目请求的 URL 是部署在 Tomcat 上的一个 Servlet，如代码 10-5 所示，对于 Get 请求和 Post 请求分别做了不同处理。关于 Servlet 开发的相关知识请查阅其他资料。

代码 10-5

```
public class LoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setCharacterEncoding("UTF - 8");
        String ln = request.getParameter("loginName");
        String lp = request.getParameter("loginPwd");
        System.out.println("服务器端,doPost 执行.登录名: 密码 -->" + ln + ":" + lp);

        response.setContentType("text/html");
        PrintWriter out  =  response.getWriter();
```

```
            out.println("服务器端,doGet 执行.登录名: 密码 -->" + ln + ":" + lp);
            out.flush();
            out.close();
        }
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            response.setCharacterEncoding("UTF-8");
            String ln = request.getParameter("loginName");
            String lp = request.getParameter("loginPwd");
            System.out.println("服务器端,doPost 执行.登录名: 密码 -->" + ln + ":" + lp);

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("服务器端,doPost 执行.登录名: 密码 -->" + ln + ":" + lp);
            out.flush();
            out.close();
        }
    }
```

10.2.4 HttpClient

使用 URLConnection 或 HttpURLConnection 可以实现访问网络资源的功能,但如果需要访问一些受保护的资源时(登录后才能访问),处理起来相当麻烦,对于这种应用场景就需要用到 HttpClient。HttpClient 位于 org.apache.http.client 包中,是 Apache 开源组织提供的项目,用于处理客户端与服务器的连接,其重要任务是处理发送 HTTP 请求,接收 HTTP 响应,管理连接,但是不会对收到的内容进行解析(浏览器的任务)。

Android 操作系统完全集成了 HttpClient,可以直接使用。相对于 10.2.3 节的内容,HttpClient 在请求资源时显得比较麻烦,除了 HttpClient 之外,还涉及其他一些类的使用。使用 HttpClient 发起 Get 请求的基本步骤如下。

(1) DefaultHttpClient client = new DefaultHttpClient(),建立 HttpClient。由于 HttpClient 是一个接口,所以在创建时,只能选择它的实现类 DefaultHttpClient 或 AndroidHttpClient。

(2) HttpGet get = new HttpGet(String url),发起 Get 请求的类。Get 请求的参数可以直接连接在 URL 后面,也可以通过方法 setParams(HttpParams params)实现。

(3) HttpResponse response = client.execute(get),请求后的响应对象,可以通过该对象获取服务器响应的信息。此处 execute(HttpUriRequest request)的参数是 HttpUriRequest 类型,HttpGet 和 HttpPost 都实现了该接口。

(4) int code = response.getStatusLine().getStatusCode(),获取服务器返回的响应代码。实际上 getStatusLine()方法会返回 StatusLine 对象,通过该对象的方法可以查看协议版本、响应代码等,常用的就是查看响应代码。HttpResponse 对象还提供了 getAllHeaders()和 getHeaders(String name)这样的方法用于查询头部信息。

(5) InputStream in = response.getEntity().getContent(),获取返回内容。方法

getEntity()会返回 HttpEntity 对象,该对象封装了服务器的响应内容。通过 HttpEntity 类的 getContent()方法可以获取 InputStream,读取返回的内容。

使用 HttpClient 发起 Post 请求的基本步骤如下,与 Get 方式稍有区别的是对请求参数的处理不同,Post 方式可以将请求参数封装成 NameValuePair 对象。

(1) DefaultHttpClient client = new DefaultHttpClient(),创建 HttpClient 对象。

(2) BasicNameValuePair pair = new BasicNameValuePair(String name, String value),创建键值对,可以是请求的头部信息,也可以是请求的参数。由于一次请求的参数对有多个,因此这些对象应该添加到一个集合中。

(3) UrlEncodedFormEntity entity = new UrlEncodedFormEntity(List<NameValuePair> list, String encoding),创建 HttpEntity 实体,UrlEncodedFormEntity 实现了 HttpEntity 接口,在创建时可以指定编码格式。

(4) HttpPost post = new HttpPost(String url),创建 HttpPost,此处 url 不包括请求串信息。

(5) post.setEntity(entity),设置请求参数。

(6) HttpResponse response = client.execute(post),执行 Post 请求,得到 HttpResponse 对象。后面的处理过程与 Get 方式一致。

(7) int code = response.getStatusLine().getStatusCode(),获取相应代码。

(8) InputStream in = response.getEntity().getContent(),读取服务器返回的数据。

新建项目 part10_4,模拟登录系统后访问其他资源的功能。activity_main.xml 是 MainActivity(代码 10-6)的布局文件,"登录系统"按钮会打开登录对话框,"查看员工"按钮直接访问受保护资源(需要登录,登录后会在 Session 中加入登录信息),TextView 用于显示操作过程。Loginlayout.xml 是登录对话框的布局文件,采用常规的登录布局方法。项目 part10_4_4 是 Web 项目,模拟服务器端程序,采用 MyEclipse 9 开发,主要有两个 Servlet,LoginServlet 用于登录,登录成功后会在 Session 中加入登录信息;EmployeeServlet 通过判断 Session 中是否有登录信息响应不同的逻辑。

代码 10-6

```
public class MainActivity extends Activity implements OnClickListener{
    Button b1,b2;
    TextView tv;
    HttpClient httpClient;
    Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            tv.append(msg.obj.toString() + "\n");
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
            b1 = (Button) findViewById(R.id.bt_login);
            b2 = (Button) findViewById(R.id.bt_show);
            tv = (TextView) findViewById(R.id.tv);
            b1.setOnClickListener(this);
            b2.setOnClickListener(this);
            //1.创建 HttpClient
            httpClient = new DefaultHttpClient();
        }
        @Override
        public void onClick(View v) {
            if(v.getId() == R.id.bt_login){
                login();
            }else if(v.getId() == R.id.bt_show){
                showEmployee();
            }
        }
        //系统登录,即访问 http://192.168.0.10:8080/part10_4_4/servlet/LoginServlet
        public void login(){
            //使用自定义对话框登录
             final View loginView = LayoutInflater.from(this).inflate(R.layout.loginlayout,
    null);
            Builder builder  = new Builder(this);
            builder.setTitle("系统登录")
            .setView(loginView)
            .setPositiveButton("登录", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                     final String ln = ((EditText)loginView.findViewById(R.id.login_name)).
    getText().toString();
                     final String lp = ((EditText)loginView.findViewById(R.id.login_pwd)).
    getText().toString();
                    //开启线程登录
                    new Thread(new Runnable(){
                        @Override
                        public void run() {
                            String urlStr = "http://192.168.0.10:8080/part10_4_4/servlet/
    LoginServlet";
                        //2.创建键值对请求参数
                        List<NameValuePair> params = new ArrayList<NameValuePair>();
                        params.add(new BasicNameValuePair("loginName",ln));
                        params.add(new BasicNameValuePair("loginPwd",lp));
                        try {
                            //3.创建实体对象
                            HttpEntity entity = new UrlEncodedFormEntity(params,HTTP.UTF_8);
                            //4.创建 HttpPost 对象
                            HttpPost post = new HttpPost(urlStr);
                            //5.设置请求参数
                            post.setEntity(entity);
                            //6.获取 HttpResponse 对象
                            HttpResponse httpResponse = httpClient.execute(post);
```

```
                        //7.获取返回信息
                        int code = httpResponse.getStatusLine().getStatusCode();
                        Log.i("Tag", "服务器返回代码 " + code);
                        String msg;
                        if(code == 200){
                            //8.读取数据,EntityUtils 是工具类,可以读取 Entity 中字符
串内容
                            msg = EntityUtils.toString(httpResponse.getEntity(), HTTP
.UTF_8);
                        }else{
                            msg = "访问服务器错误!";
                        }
                        //Handler 通知主 UI 更新
                        Message message = new Message();
                        message.obj = msg;
                        handler.sendMessage(message);
                    } catch (UnsupportedEncodingException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } catch (ClientProtocolException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }).start();
        }
    })
    .setNegativeButton("取消", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

        }
    }).show();
}
//查看受保护的资源,即访问 http://192.168.0.10:8080/part10_4_4/servlet/EmployeeServlet
//如果没有登录,Session 中无登录信息,将无法查看
public void showEmployee(){
    new Thread(new Runnable(){
        @Override
        public void run() {
            String urlStr = "http://192.168.0.10:8080/part10_4_4/servlet/
EmployeeServlet";
            //2.创建 HttpGet
            HttpGet httpGet = new HttpGet(urlStr);
            BufferedReader br = null;
            try {
```

```
                        //3.获取 HttpResponse 对象
                        HttpResponse httpResponse = httpClient.execute(httpGet);
                        //4.查看信息
                        int code = httpResponse.getStatusLine().getStatusCode();
                        Log.i("Tag", "服务器返回代码 " + code);
                        String msg;
                        if(code == 200){
                            //5.读取数据
                            HttpEntity entity = httpResponse.getEntity();
                            br = new BufferedReader(new InputStreamReader(entity.getContent()));
                            msg = br.readLine();
                        }else{
                            msg = "访问服务器错误!";
                        }
                        //Handler 通知主 UI 更新
                        Message message = new Message();
                        message.obj = msg;
                        handler.sendMessage(message);
                    } catch (ClientProtocolException e) {
                        // TODO Auto - generated catch block
                        e.printStackTrace();
                    } catch (IOException e) {
                        // TODO Auto - generated catch block
                        e.printStackTrace();
                    }finally{
                        if(br!= null)
                            try {
                                br.close();
                            } catch (IOException e) {
                                // TODO Auto - generated catch block
                                e.printStackTrace();
                            }
                    }
                }
            }).start();
        }
    }
```

服务器端项目需要部署在服务器(如 Tomcat),手机端才能正常访问。如果直接单击“查看员工”按钮,需要登录才可以查看的内容是无法看到的,提示“请先登录”(该提示由 EmployeeServlet 返回)。成功登录后,提示“登录成功……”(该提示由 LoginServlet 返回),再次单击“查看员工”按钮,提示“欢迎……”(该提示由 EmployeeServlet 返回),项目运行效果如图 10.3 和图 10.4 所示。

代码 10-6 中系统登录采用 Post 方式,查看员工信息采用 Get 方式。两种 HTTP 请求的实现步骤在代码中已经做了详细注释(黑体部分)。总体而言,采用 HttpClient 访问网络资源所能实现的功能,比直接使用 URLConnection 更加全面,但这种方式涉及的类比较多,需要多加练习。按照前面给出的步骤,基本能够完成常规资源的访问。



图 10.3　登录界面

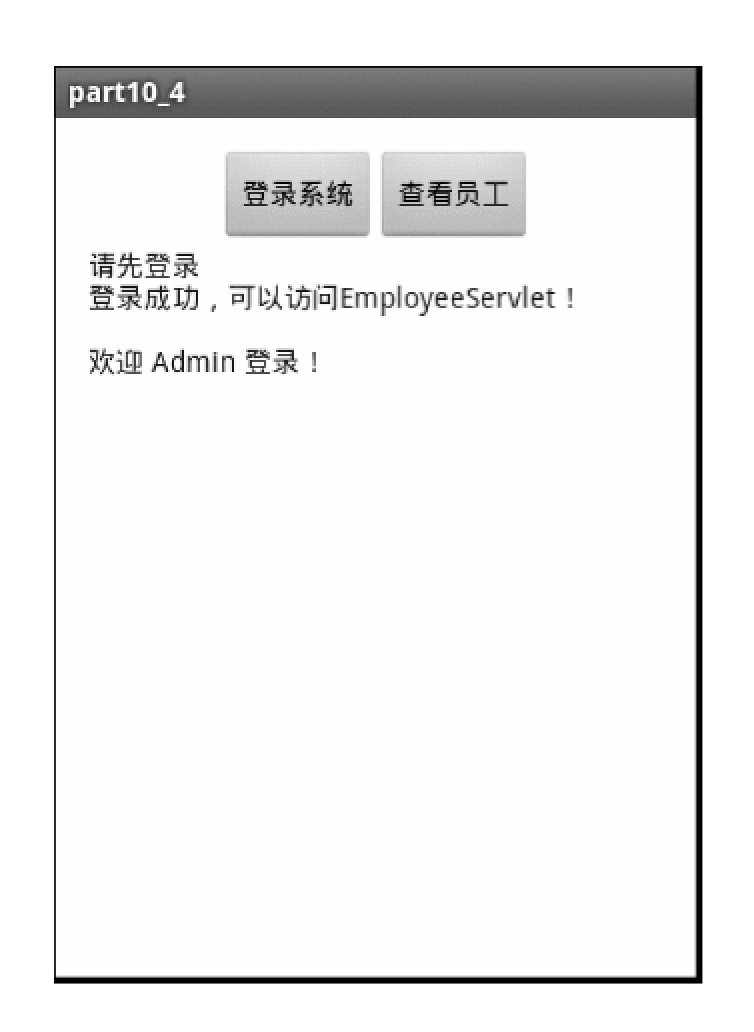


图 10.4　登录后可以访问受保护内容

10.3　使用 Web Service

Web Service 是一种基于 SAOP 的远程调用标准，通过它可以将不同操作系统平台、不同语言、不同技术整合到一起；也是一种基于 Web 的服务，也是一个应用程序，它向外界暴露出一个能够通过 Web 进行调用的 API，通过编程的方法就可以调用这个应用程序。一般把能调用这个 Web Service 的应用程序叫作客户端程序。本节主要讨论如何在 Android 中调用 Web Service，实现一些比较实用的功能、如天气预报功能、票务查询功能等。如果对发布 Web Service 感兴趣可以查阅其他资料。

10.3.1　调用 Web Service

在 Android 平台调用 Web Service 需要依赖于第三方类库 ksoap2，它是一个 SOAP Web Service 客户端开发包，主要用于资源受限制的 Java 环境，如 Applets 或 J2ME 应用程序(CLDC/ CDC/MIDP)。在 Android 平台中并不会直接使用 ksoap2，而是使用 ksoap2 Android。ksoap2 Android 是 Android 平台上一个高效、轻量级的 SOAP 开发包，等同于 Android 平台上的 ksoap2 的移植版本。

ksoap2 Android 当前的最新版本为 3.0.0，名为 ksoap2-android-assembly-3.0.0-jar-with-dependencies.jar，它的下载地址是 https://code.google.com/p/ksoap2-android/wiki/HowToUse? tm=2。对于联网下载不便的读者可以直接使用项目 part10_5 中的 jar 文件。

在 Android 项目中添加 jar 文件与 Java 项目中添加 jar 文件一样，先在项目中建立 libs 文件夹(ADT 4.2 以后创建的 Android 项目会自动生成 libs 文件夹)，把 ksoap2-android-assembly-3.0.0-jar-with-dependencies.jar 复制到 libs，然后在项目上右击，选择 Builder Path→Config Builde Path 命令打开如图 10.5 所示的窗口，切换到 Libraries 选项卡，将刚才

复制到 libs 文件夹中的 JAR 文件添加进引用类库。

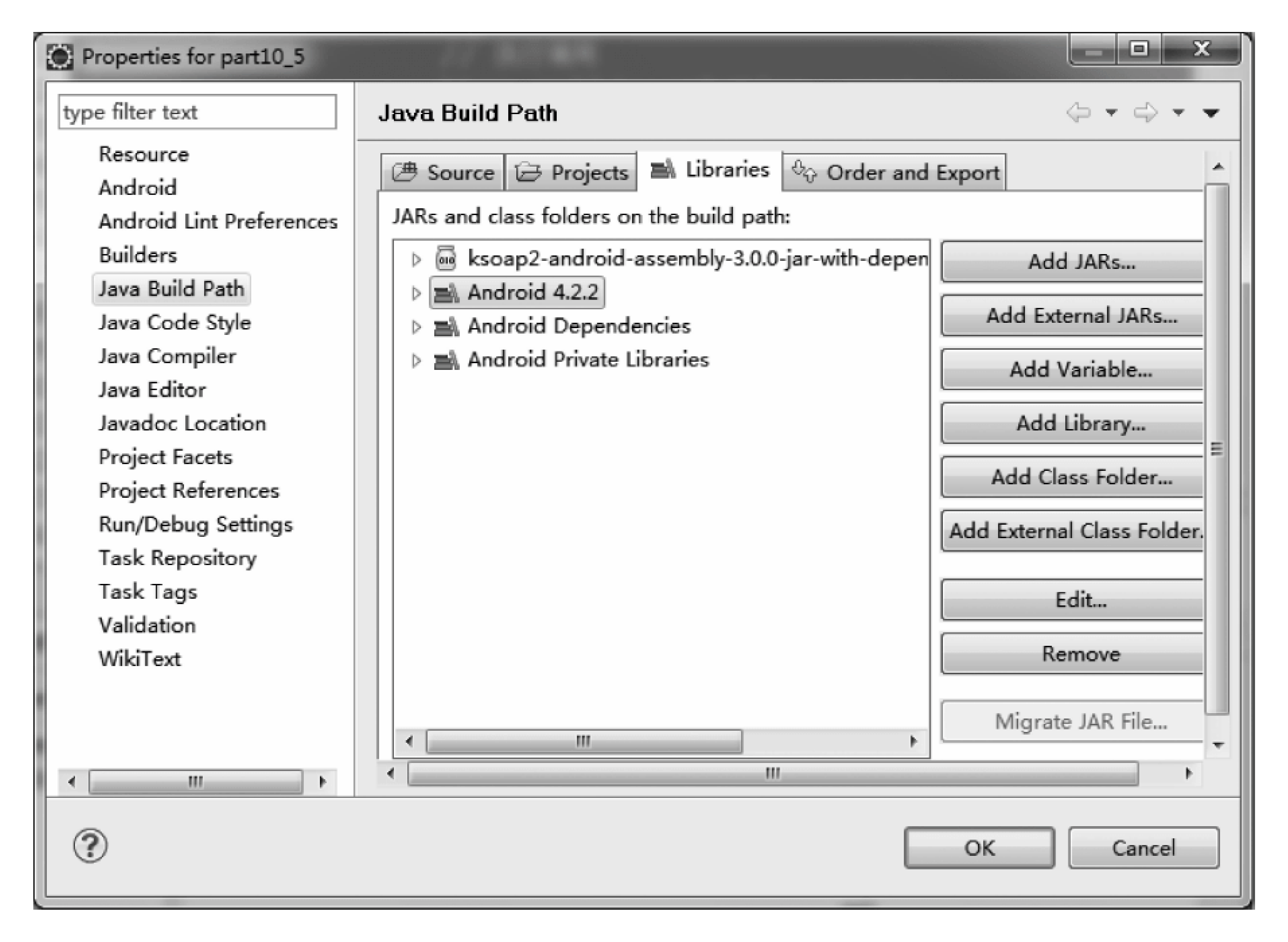


图 10.5　添加 JAR 文件

在 Android 中使用 ksoap2 Android 调用 Web Service 的步骤如下。

（1）创建 HttpTransportSErvice 对象，通过 HttpTransportSE 类的构造方法可以指定 Web Service 的 WSDL（Web Service 描述语言）文档的 URL，该 URL 由 Web Service 提供。图 10.6 是 Web XML 网站提供的手机号码归属地查询的 Web Service 描述。

（2）创建 SoapSerializationEnvelope 对象，通过构造方法设置 SOAP 的版本号（一般为 SoapEnvelope.VER11）。该版本号需要根据服务端 Web Service 的版本号设置。在创建 SoapSerializationEnvelope 对象后，还需要设置 SoapSerializationEnvelope 类的 bodyOut 属性。

（3）创建 SoapObject 对象，该类构造方法的第一个参数表示 Web Service 的命名空间，可以从 WSDL 文档中找到，第二个参数表示要调用的 Web Service 方法名。



图 10.6　Web XML 网站手机号码查询 Web 服务

（4）如果需要给 Web Service 传送参数（一般情况都需要），需要使用 SoapObject 类的 addProperty(String key，Object value)方法，其中 key 是 Web Service 要求的参数，不能随意写。

(5) 调用 SoapSerializationEnvelope 类的 setOutputSoapObject()方法或者直接使用 bodyOut 属性,将 SoapObject 对象附加给 SoapSerializationEnvelope,以明确请求参数。

(6) 调用 HttpTransportSErvice 的 call 方法,执行 Web Service 访问。call 方法有两个参数,第一是 Web Service 的 SOAPAction(这可以从 Web Service 提供的文档找到,如代码 10-7 所示,Web XML 网站提供的用于手机号码查询 SOAP 的描述信息),第二个是 SoapSerializationEnvelope 对象。

(7) 获取返回的数据,通过 SoapSerializationEnvelope 对象的 bodyIn 属性可以获取 SoapObject 对象,该对象是 Web Service 返回的信息,解析该对象就能得到返回值。

代码 10-7

```
POST /WebServices/MobileCodeWS.asmx HTTP/1.1
Host: webservice.webxml.com.cn
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://WebXml.com.cn/getMobileCodeInfo"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfo xmlns="http://WebXml.com.cn/">
      <mobileCode>string</mobileCode>
      <userID>string</userID>
    </getMobileCodeInfo>
  </soap:Body>
</soap:Envelope>
```

新建项目 part10_5,通过调用 Web XML 网站提供的 Web Service,实现在手机端查询手机号码归属地的功能。布局文本比较简单,一个文本框用于输入手机号码,一个按钮触发查询。MainActivity.java 的代码如代码 10-8 所示。

代码 10-8

```
public class MainActivity extends Activity {
    EditText et;
    Button bt;
    TextView tv;
    Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            tv.append(msg.obj.toString() + "\n");
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText) findViewById(R.id.et);
        bt = (Button) findViewById(R.id.bt);
        tv = (TextView) findViewById(R.id.tv);
        bt.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View v) {
                //验证手机号码的合法性,略
                String p = et.getText().toString().trim();
                Log.i("Tag", p);
                //if(p.length()<11)return;
                getPhoneWebService(p);
            }}
        );
    }
    //调用 Web Service 查看手机所属地
    public void getPhoneWebService(final String phone) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // Web Service,以下信息由 Web Service 提供
                 String surl  = "http://webservice.webxml.com.cn/WebServices/MobileCodeWS.
asmx";
                String ns  =  "http://WebXml.com.cn/";
                String method = "getMobileCodeInfo";
                final String soapAction  = "http://WebXml.com.cn/getMobileCodeInfo";
                //1.创建 HttpTransportSErvice,用于调用 Web Service
                final HttpTransportSE htse  =  new HttpTransportSE(surl);
                //2.生成调用 Web Service 方法的 SOAP 请求信息,并指定 SOAP 的版本
                final SoapSerializationEnvelope sse  =  new SoapSerializationEnvelope(
                        SoapEnvelope.VER11);
                sse.dotNet = true;
                //3.创建 SoapObject,可以传进参数,以下两个参数是 Web Service 要求的
                SoapObject so  =  new SoapObject(ns, method);
                so.addProperty("mobileCode", phone);
                so.addProperty("userID", "");
                Log.i("Tag", so.toString());
                //4.设置 SOAP 的传出消息体
                sse.bodyOut  =  so;
                //5.调用 Web Service,开启线程
                try {
                    // 执行调用
                    htse.call(soapAction, sse);
                    //6.处理返回结果
                    String result;
                    if (sse.getResponse() != null) {
                        // 获取返回的数据
                        SoapObject object  =  (SoapObject) sse.bodyIn;
                        Log.i("Tag", object.toString());
```

```
                    // 获取返回的结果
                    result = object.getProperty(0).toString();
                } else {
                    result = "WebService 调用无返回值!";
                }
                Message msg = new Message();
                msg.obj = result;
                handler.sendMessage(msg);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (XmlPullParserException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }).start();
  }
}
```

在代码 10-8 中，创建以及使用 ksoap2 Android 调用 Web Service 的步骤已经用黑体表示。所用到的信息(如 url、ns、methed 等)都可以在 Web Service 上查找到。所需设置的参数 so.addProperty("mobileCode", phone)和 so.addProperty("userID", "")都可以在代码 10-8 的描述中找到。

项目运行前需要开启联网权限，输入手机号码就可以查询该号码的归属地，运行效果如图 10.7 所示。

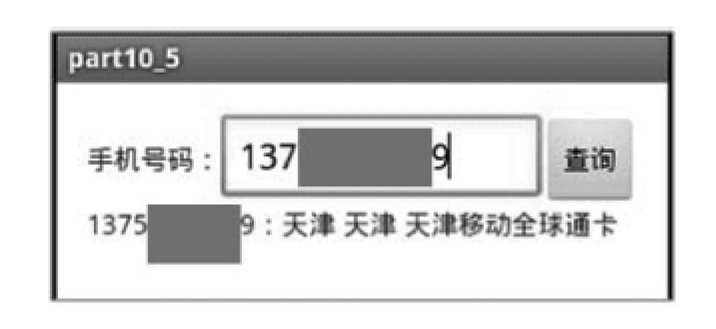


图 10.7　调用 Web Service 运行效果

10.3.2　解析 XML

XML(eXtensible Markup Language，可扩展标记语言)是一种标记语言。XML 设计用来传送及携带数据信息，不用来表现或展示数据，HTML 则用来表现数据，所以 XML 用途的焦点是它说明数据是什么，以及携带数据信息。由于 XML 数据以纯文本格式进行存储，因此提供了一种独立于软件和硬件的数据存储方法，让不同应用程序共享数据变得更加容易。关于 XML 的知识可以查阅其他资源，本节主要讨论在 Android 系统中如何解析收到的 XML 数据。

在 Android 系统中，常见的 XML 解析器有 DOM 解析器、SAX 解析器和 PULL 解析器。

DOM(Document Object Model)是基于树状结构的节点或信息片段的集合，可以使用

DOM API 遍历 XML 树、检索所需数据。分析该结构通常需要加载整个文档和构造树状结构，然后才可以检索和更新节点信息。Android 完全支持 DOM 解析。由于 DOM 在内存中以树状结构存放，因此检索和更新效率会更高。但是对于特别大的文档，解析和加载整个文档将会很耗资源。

SAX(Simple API for XML)解析器是一种基于事件的解析器，从文件的开始顺序解析到文档的结束，不可暂停或倒退。它的核心是事件处理模式，主要是围绕着事件源以及事件处理器来工作的。SAX 的工作原理简单地说就是对文档进行顺序扫描，当扫描到文档(document)开始与结束、元素(element)开始与结束、文档(document)结束等地方时通知事件处理函数，由事件处理函数做相应动作，然后继续同样的扫描，直至文档结束。SAX 解析器的优点是解析速度快，占用内存少，非常适合在 Android 移动设备中使用。

PULL 解析器是 Android 附带的解析器，其工作方式类似于 SAX。允许应用程序代码从解析器中获取事件，PULL 解析器的运行方式和 SAX 类似，都是基于事件的模式。PULL 解析器小巧轻便，解析速度快，简单易用，非常适合在 Android 移动设备中使用。Android 系统内部在解析各种 XML 时也是用 PULL 解析器，Android 官方推荐开发者们使用 PULL 解析技术。PULL 解析技术是第三方开发的开源技术，它同样可以应用于 JavaSE 开发。本节主要介绍 PULL 解析技术的实现方式。

Android 系统中和 PULL 方式相关的包是 org.xmlpull.v1，在这个包中提供了 PULL 解析器的工厂类 XmlPullParserFactory 和 PULL 解析器 XmlPullParser。XmlPullParserFactory 对象通过调用 newPullParser 方法创建 XmlPullParser 解析器对象。

在开发过程中可以通过以下两种方式创建 XmlPullParser 对象。

1. 使用工厂类创建

```
XmlPullParserFactory pullFactory = XmlPullParserFactory.newInstance();
XmlPullParser xmlPullParser = pullFactory.newPullParser();
```

2. 使用 Android 提供的实用工具类 android.util.Xml 创建

```
XmlPullParser xmlPullParser = Xml.newPullParser();
```

创建 XmlPullParser 对象之后，可以调用 setInput(InputStream inputStream, String inputEncoding)方法，设置需要解析的文件流，然后调用 getEventType()方法获取元素，通过判断元素是否是 START_DOCUMENT、END_DOCUMENT、START_TAG、END_TAG、TEXT，进行相应解析。PULL 方式比较简单，而且可以根据判断停止解析(DOM 和 SAX 都需要对文件的文章解析中途不能停止)。

10.3.3 航班信息查询

在 Android 平台实现航班信息查询的功能需要使用第三方 Web Service，以获取详细的航班信息。下面介绍使用 Web XML 网站提供的航班信息查询服务，实现航班信息查询功

能。在使用第三方 Web Service 之前，需要仔细了解使用文档的说明，弄清调用 Web Service 的方式。比如 Web XML 所提供的方式有以下三种。

采用 SOAP 访问航班信息查询服务的描述如代码 10-9 所示。

代码 10-9

```
POST /webservices/DomesticAirline.asmx HTTP/1.1
Host: webservice.webxml.com.cn
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://WebXml.com.cn/getDomesticAirlinesTime"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getDomesticAirlinesTime xmlns="http://WebXml.com.cn/">
      <startCity>string</startCity>
      <lastCity>string</lastCity>
      <theDate>string</theDate>
      <userID>string</userID>
    </getDomesticAirlinesTime>
  </soap:Body>
</soap:Envelope>
```

采用 Get 方式调用航班信息查询服务的描述如代码 10-10 所示。

代码 10-10

```
GET /webservices/DomesticAirline.asmx/getDomesticAirlinesTime
              ?startCity=string&lastCity=string&theDate=string&userID=string HTTP/1.1
Host: webservice.webxml.com.cn
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

采用 Post 方式调用航班信息查询服务的描述如代码 10-11 所示。

代码 10-11

```
POST /webservices/DomesticAirline.asmx/getDomesticAirlinesTime HTTP/1.1
Host: webservice.webxml.com.cn
Content-Type: application/x-www-form-urlencoded
Content-Length: length

startCity=string&lastCity=string&theDate=string&userID=string
```

新建项目 part10_6，演示使用 Get 方式访问 Web Service 获取航班信息，完整代码请查阅随书配套资料，下面仅介绍如何解析获取的 XML 文件。在 Web XML 网站调用 Web

Service 需要注册，根据不同的服务开启相应的使用请求即可。在该项目中，为了能够更加直观地查看从 Web Service 获取的数据，将查询到的航班信息存到了 SD 卡，将该文件复制到计算机，打开文件后，显示如代码 10-12 所示的格式。

该文件是 XML 格式的文件，只要解析该 XML 文件就可以获取到相应的航班信息，然后呈现到列表控件中，就可以实现航班信息查询功能。

代码 10-12

```
<?xml version = "1.0" encoding = "utf - 8"?>
<DataSet xmlns = "http://WebXml.com.cn/">
  <xs:schema id = "Airlines" xmlns = "" xmlns:xs = "http://www.w3.org/2001/XMLSchema"
                          xmlns:msdata = "urn:schemas - microsoft - com:xml - msdata">
    <xs:element name = "Airlines" msdata:IsDataSet = "true" msdata:UseCurrentLocale = "true">
      <xs:complexType>
        <xs:choice minOccurs = "0" maxOccurs = "unbounded">
          <xs:element name = "AirlinesTime">
            <xs:complexType>
              <xs:sequence>
                <xs:element name = "Company" type = "xs:string" minOccurs = "0" />
                <xs:element name = "AirlineCode" type = "xs:string" minOccurs = "0" />
                <xs:element name = "StartDrome" type = "xs:string" minOccurs = "0" />
                <xs:element name = "ArriveDrome" type = "xs:string" minOccurs = "0" />
                <xs:element name = "StartTime" type = "xs:string" minOccurs = "0" />
                <xs:element name = "ArriveTime" type = "xs:string" minOccurs = "0" />
                <xs:element name = "Mode" type = "xs:string" minOccurs = "0" />
                <xs:element name = "AirlineStop" type = "xs:string" minOccurs = "0" />
                <xs:element name = "Week" type = "xs:string" minOccurs = "0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata = "urn:schemas - microsoft - com:xml - msdata" xmlns:diffgr =
"urn:schemas - microsoft - com:xml - diffgram - v1">
    <Airlines xmlns = "">
      <AirlinesTime diffgr:id = "AirlinesTime1" msdata:rowOrder = "0">
        <Company>海南航空</Company>
        <AirlineCode>HU7205 </AirlineCode>
        <StartDrome>天津滨海国际机场</StartDrome>
        <ArriveDrome>上海浦东国际机场</ArriveDrome>
        <StartTime>07:55 </StartTime>
        <ArriveTime>10:00 </ArriveTime>
        <Mode>738 </Mode>
        <AirlineStop>0 </AirlineStop>
        <Week>123456 日</Week>
      </AirlinesTime>
      <AirlinesTime diffgr:id = "AirlinesTime2" msdata:rowOrder = "1">
```

```
            <Company>天津航空</Company>
            <AirlineCode>GS7205</AirlineCode>
            <StartDrome>天津滨海国际机场</StartDrome>
            <ArriveDrome>上海浦东国际机场</ArriveDrome>
            <StartTime>07:55</StartTime>
            <ArriveTime>10:00</ArriveTime>
            <Mode>738</Mode>
            <AirlineStop>0</AirlineStop>
            <Week>123456 日</Week>
        </AirlinesTime>
        <AirlinesTime diffgr:id = "AirlinesTime3" msdata:rowOrder = "2">
            <Company>东方航空</Company>
            <AirlineCode>MU9070</AirlineCode>
            <StartDrome>天津滨海国际机场</StartDrome>
            <ArriveDrome>上海浦东国际机场</ArriveDrome>
            <StartTime>08:00</StartTime>
            <ArriveTime>09:50</ArriveTime>
            <Mode>738</Mode>
            <AirlineStop>0</AirlineStop>
            <Week>245 日</Week>
        </AirlinesTime>
      其他航班信息  略
      </Airlines>
    </diffgr:diffgram>
</DataSet>
```

在该项目中，XML 文件的解析是通过类 MyPullXmlParser 实现的，如代码 10-13 所示。将解析到的每个航班信息都存入 Map，再将 Map 放入 List。解析过程比较直观，通过判断 XML 文件中的标签执行不同逻辑，代码中已经给出解释。

代码 10-13

```
public class MyPullXmlParser {
    MainActivity act;
    public MyPullXmlParser(MainActivity ma){
        act = ma;
    }
    public List<Map<String,String>> parseXml(InputStream is){
        List<Map<String,String>> lines = null;
        if(is == null)return null;
        //创建解析
        XmlPullParser parser = Xml.newPullParser();
        try {
            lines  = new ArrayList<Map<String,String>>();
            parser.setInput(is, "UTF-8");
            //获取标签名
            int type = parser.getEventType();
            Map<String,String> al = null;
```

```
int n = 0;
while((type = parser.next())!= XmlPullParser.END_DOCUMENT){
    n++;
    String tag = parser.getName();
    Log.i("Tag", "tag = " + tag + "\t type" + type);
    //当检测到新航班标签时,创建航班对象
    if(type == XmlPullParser.START_TAG&&tag.equalsIgnoreCase("AirlinesTime")){
        //航线对象
         al = new HashMap<String,String>();
    }
    if(type == XmlPullParser.START_TAG){
        //设置航班参数
        if(tag.equalsIgnoreCase("Company")){
            type = parser.next();
            al.put("company",parser.getText());
        }else if(tag.equalsIgnoreCase("AirlineCode")){
            type = parser.next();
            al.put("code",parser.getText());
        }else if(tag.equalsIgnoreCase("StartDrome")){
            type = parser.next();
            al.put("startDrome",parser.getText());
        }else if(tag.equalsIgnoreCase("ArriveDrome")){
            type = parser.next();
            al.put("arriverDrome",parser.getText());
        }else if(tag.equalsIgnoreCase("StartTime")){
            type = parser.next();
            al.put("startTime",parser.getText());
        }else if(tag.equalsIgnoreCase("ArriveTime")){
            type = parser.next();
            al.put("arriveTime", parser.getText());
        }else if(tag.equalsIgnoreCase("Mode")){
            type = parser.next();
            al.put("mode", parser.getText());
        }else if(tag.equalsIgnoreCase("AirlineStop")){
            type = parser.next();
            al.put("stop", parser.getText());
        }else if(tag.equalsIgnoreCase("Week")){
            type = parser.next();
            al.put("week", parser.getText());
        }
    }
    if(type == XmlPullParser.END_TAG &&tag.equalsIgnoreCase("AirlinesTime")){
        //航班信息解析完整,加入集合
        lines.add(al);
    }
    //更改进度条
    if(n % 50 == 0)
    act.handler.post(new Runnable(){
        @Override
        public void run() {
```

```
                        act.progress.setProgress(act.progress.getProgress() + 1);
                    }
                });
            }
        } catch (XmlPullParserException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto - generated catch block
            e.printStackTrace();
        }
        Log.i("Tag", "解析结束,共有对象各数" + lines.size());
        return lines;
    }
}
```

在运行该项目时注意加入相应权限,查询效果如图 10.8 和图 10.9 所示。在出发城市和目的城市出入任意有民航机场的城市,设定时间,单击"查询"按钮就可以在 Web Service 获取到航班信息,解析 XML 文件,呈现到 ListView 控件即可。



图 10.8　等待解析航班信息



图 10.9　显示航班信息界面

10.3.4　解析 JSON

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式,常用于 JavaScript 语言的数据交换,现在已经逐渐成为一种语言无关的数据交换格式,这一点类似于 XML,但它要比 XML 更加轻量级。

Android 操作系统集成了对 JSON 的支持,与 JSON 解析相关的类位于 org.json 包中,主要有 JSONArray(数组形式的,数组元素可以是对象)、JSONObject(对象形式)、JSONString 等。通过这些类就可以很轻松地实现 JSON 字符串与 JSONObject 和

JSONArray 直接的相互转换。

新建项目 part10_7，采用 HttpClient 访问 Web Servlet，获取 JSON 字符串，然后封装成 Java 对象。Web Servlet 使用 MyEclipse 9 开发，完整项目请查阅 part10_7_7 项目，在运作之前需要部署到服务器。

代码 10-14

```
public class MainActivity extends Activity implements OnClickListener{
    EditText et;
    Button bt;
    TextView tv;
    HttpClient httpClient = new DefaultHttpClient();
    Handler handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            tv.append(msg.obj.toString() + "\n");
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText) findViewById(R.id.et);
        bt = (Button) findViewById(R.id.bt);
        tv = (TextView) findViewById(R.id.tv);
        bt.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        String id = et.getText().toString();
        if(id.length()> 0)getBookFromNet(id);
    }
    public void getBookFromNet(final String id){
        final String urlStr = "http://192.168.0.10:8080/part10_7_7/servlet/GetBook";
        new Thread(new Runnable(){
            @Override
            public void run() {
                HttpPost post = new HttpPost(urlStr);
                List<NameValuePair> params = new ArrayList<NameValuePair>();
                params.add(new BasicNameValuePair("id",id));
                try {
                    post.setEntity(new UrlEncodedFormEntity(params,HTTP.UTF_8));
                    HttpResponse httpResponse = httpClient.execute(post);
                    int code = httpResponse.getStatusLine().getStatusCode();
                    Log.i("Tag", "服务器返回码 " + code);
                    String msg;
                    if(code == 200){
                        //读取数据,EntityUtils 是工具类,可以读取 Entity 中字符串内容
```

```
                        msg = EntityUtils.toString(httpResponse.getEntity(), HTTP.UTF_8);
                        Log.i("Tag", "未经解析的服务器返回的 Json 字符串 " + msg);
                        //Handler 通知主 UI 更新
                        Message message = new Message();
                        message.obj = msg;
                        handler.sendMessage(message);
                        //将收到的 JSON 串转换为 Java 对象
                        JSONArray jsonArray = new JSONArray(msg);
                        //因为只有一个对象
                        JSONObject jsonObj = jsonArray.getJSONObject(0);
                        Book book = new Book();
                        book.setId(jsonObj.getString("id"));
                        book.setBookName(jsonObj.getString("bookName"));
                        book.setBookPrice(jsonObj.getString("bookPrice"));
                        Log.i("Tag", "已经解析为 Book 对象的字符串 " + book.toString());
                        //Handler 通知主 UI 更新
                        Message message2 = new Message();
                        message2.obj = book.toString();
                        handler.sendMessage(message2);
                    }else{
                        msg = "访问服务器错误!";
                        //Handler 通知主 UI 更新
                        Message message = new Message();
                        message.obj = msg;
                        handler.sendMessage(message);
                    }
                } catch (UnsupportedEncodingException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (ClientProtocolException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (JSONException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

项目运行效果如图 10.10 所示，上面的字符串是服务器端直接返回的标准 JSON 文本，下面的字符串是封装为 Book 对象之后的字符串文本。将 JSON 文本解析为 JSONArray，可以包含多个 JSONObject 对象，该项目中服务器端只返回了一个对象，所以代码中使用 jsonArray.getJSONObject(0)获取该对象。得到 JSONObject 对象之后，就可以获取该对象的一系列属性了。

将 Java 对象转换为 JSON 文本的方式相对比较简单，如代码 10-15 所示，只需要调用 JSONArray 的构造方法创建 JSONArray 对象，然后 toString 即可。特别注意的是，JSONArray 构造方法的参数只能是 List 集合、数组、字符串等，不能传入 Map 集合。

代码 10-15

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setCharacterEncoding("UTF-8");
    response.setContentType("text/html");
    String id = request.getParameter("id");
    List<Book> bs  = new ArrayList<Book>();
    bs.add(books.get(id));
    //将 Java 对象转换为 JSON 串
    JSONArray jsonArray = new JSONArray(bs);
    String jsonStr;
    jsonStr = jsonArray.toString();
    System.out.println(jsonStr);
    PrintWriter out = response.getWriter();
    out.println(jsonStr);
    out.flush();
    out.close();
}
```



图 10.10　JSON 请求结果

在 Android 系统中进行网络编程有很多种方式，在不同的场合下会有区别，请根据需要选择较为恰当的网络编程方法。XML 和 JSON 都可以在不同编程语言、不同操作系统之间交换数据，这也是目前比较常用的两种获取网络数据的方式，其中 XML 方式具有严格的格式，JSON 相对更加轻量级。关于如何选择二者请考虑如下建议。

- JSON 使用格式比较简单，其主战场在 Web 开发的前端，即 JavaScript 的应用场景。XML 格式更加规范，可扩展性较强。
- JSON 和 XML 的轻/重量级的区别在于，JSON 只提供了整体解析方案，而这种方法只在解析较少的数据时才能起到良好效果；而 XML 提供了对大规模数据的逐步解析方案，这种方案很适用于对大量数据的处理。
- 传输数据较少时建议使用 JSON，当数据量较大时建议使用 XML。

习 题

1. 如何理解 Socket 编程和 TCP 通信协议的关系？
2. Get 请求与 Post 请求的区别有哪些？
3. 使用 HttpClient 如何实现 Get 请求和 Post 请求？
4. 使用 SOAP 访问 Web Service 的步骤有哪些？
5. 如何解析 XML 文件和 JSON 数据？

参考文献

[1] Cay S Horstmann,Gary Cornell. Java 核心技术卷Ⅰ、卷Ⅱ. 北京：机械工业出版社,2008.
[2] 李刚. 疯狂 Android 讲义. 北京：电子工业出版社,2013.
[3] 李华明. Android 游戏编程从零开始. 北京：清华大学出版社,2011.
[4] http://developer. android. com/develop/index. html,2013. 8.
[5] Shane Conder,Lauren Darcey. Android 移动应用开发从入门到精通. 北京：人民邮电出版社,2011.
[6] 杨云君. Android 的设计与实现:卷 1. 北京：机械工业出版社,2013.
[7] Reto Meier. Android 4 高级编程(第 3 版). 北京：清华大学出版社,2013.
[8] 范怀宇. Android 开发精要. 北京：机械工业出版社,2012.
[9] Jason Ostrander. Android UI 基础教程. 北京：人民邮电出版社,2012.
[10] Darwin. Android 应用开发攻略. 北京：机械工业出版社,2013.
[11] 朱凤山. Java 语言课程设计指导. 北京：清华大学出版社,2012.
[12] 于静. Java Web 应用开发教程. 北京：北京邮电大学出版社,2010.
[13] 孙鑫. Java Web 开发详解. 北京：电子工业出版社,2012.
[14] 陈亚辉,缪勇. SSH 框架技术与项目实战. 北京：清华大学出版社,2012.